

①⑨ BUNDESREPUBLIK  
DEUTSCHLAND



DEUTSCHES  
PATENTAMT

⑫ **Offenlegungsschrift**  
⑪ **DE 3631992 A 1**

⑤① Int. Cl. 4:  
**G 09 C 1/00**  
// G 06 F 7/50

②① Aktenzeichen: P 36 31 992.9  
②② Anmeldetag: 20. 9. 86  
④③ Offenlegungstag: 5. 11. 87

**Behördeneigentum**

DE 3631992 A 1

③⑩ Innere Priorität: ③② ③③ ③①  
05.03.86 DE 36 07 646.5

⑦① Anmelder:  
Sedlak, Holger, 3300 Braunschweig, DE

⑦④ Vertreter:  
Thömen, U., Dipl.-Ing., Pat.-Anw., 3000 Hannover

⑦② Erfinder:  
gleich Anmelder

Prüfungsantrag gem. § 44 PatG ist gestellt

⑤④ **Kryptographie-Verfahren und Kryptographie-Prozessor zur Durchführung des Verfahrens**

Mit der wachsenden Verbreitung elektronischer Verfahren der Kommunikation ist die Forderung nach Geheimhaltung der Kommunikationsdaten und der Absenderauthentizität unverzichtbar.

Die Erfindung geht aus von dem Public-Key-Code-Verfahren in seiner Realisierung nach dem RSA-Verfahren. Es wird ein Kryptographie-Prozessor geschaffen, der Daten nach dem RSA-Verfahren ver- und/oder entschlüsselt und den Anforderungen einer digitalen Schnittstelle eines ISDN-Netzes genügt.

Der Prozessor ist in VLSI-Technik aufgebaut und besitzt im Sinne einer wirtschaftlichen Anwendbarkeit nur geringe Abmessungen und ermöglicht trotzdem bisher nicht erreichte Verschlüsselungsraten.

DE 3631992 A 1

## Patentansprüche

1. Kryptographie-Verfahren nach der Public-Key-Code-Methode von Rivest, Shamir und Adleman (RSA-Verfahren), umfassend die Anwendung der nachfolgenden Operationen zur Chiffrierung bzw. Verschlüsselung von Nachrichten:

- Auswahl zweier großer Primzahlen  $p$  und  $q$  und einer weiteren großen Zahl  $E$ ,
- Bildung des Produktes  $N = p \cdot q$ ,
- Umwandlung der zu chiffrierenden Nachrichten in eine Kette von vorzugsweise gleichlangen Gliedern  $P_i$ , deren Werte als Zahl kleiner als der Wert der Zahl  $N$  ist,
- Chiffrierung dieser Glieder durch die jeweilige Erhebung in die  $E$ -te Potenz mit anschließender Bildung von Modulo  $N$ , (d. h., es entstehen die Zahlen  $C_i = p_i^E \text{ Modulo } N$ ),
- wobei die Potenzierung durch eine Folge von Multiplikationen ersetzt wird und nach jeder Multiplikation sofort eine Modulo-Operation ausgeführt wird, (d. h., es wird im Restklassenring über  $N$  multipliziert),
- wobei die Multiplikation in Einzelschritte zerlegt wird, so daß aus der Multiplikation eine Folge von Additionen entsteht, und
- wobei die Modulo-Operation nach dem klassischen Divisionsalgorithmus durch eine Folge von Subtraktionen ersetzt wird,

gekennzeichnet durch die Anwendung eines ersten Look-Ahead-Verfahrens (Vorausrechnungsverfahrens) für die Division (Fig. 5), so daß auch die Multiplikation mit einem zweiten Look-Ahead-Verfahren (Fig. 4) durchführbar ist.

2. Kryptographie-Verfahren nach Anspruch 1, gekennzeichnet durch Look-Ahead-Algorithmen, mit denen die maximal notwendige Anzahl von Additionen bzw. Subtraktionen reduziert wird.

3. Kryptographie-Verfahren nach Anspruch 2, dadurch gekennzeichnet, daß das erste Look-Ahead-Verfahren für die Modulo-Operation so gewählt ist, daß der wahrscheinlichkeitstheoretische Erwartungswert der Anzahl der beim ersten Look-Ahead-Verfahren übersprungenen Operationen genau so groß ist wie der wahrscheinlichkeitstheoretische Erwartungswert, der beim zweiten Look-Ahead-Verfahren für die Multiplikation übersprungenen Operationen.

4. Kryptographie-Verfahren nach Anspruch 3, gekennzeichnet durch eine Entkopplung der beiden Look-Ahead-Verfahren, wobei jeder der beiden Look-Ahead-Verfahren einen Schiebebetrug ( $sz$  bzw.  $sn$ ) erzeugt, der angibt, um wieviel Bits das Zwischenergebnis ( $Z$ ) der Multiplikation bzw. der Modulus ( $N$ ) pro Zyklus verschoben wird, wobei das Zwischenergebnis ( $Z$ ) absolut um  $sz$ -Bits und der Modulus ( $N$ ) relativ zum Zwischenergebnis ( $Z$ ) um  $sn$ -Bits verschoben wird.

5. Kryptographie-Verfahren nach einem der vorhergehenden Ansprüche 1—4, gekennzeichnet durch die Zusammenfassung der Addition bzw. Subtraktion des Multiplikations- und des Modulo-Schrittes zu einer einzigen Operation (3-Operanden-Addition), wobei pro Schritt nicht zwei, sondern drei Operanden wie folgt addiert werden:

$$\begin{array}{r} A[i] \\ + B[i] \\ + C[i] \\ \hline S1 \quad S0 = 0 \dots 3 \end{array}$$

$$X[\max + 1] \dots X[i + 1] \dots X[0] = 0$$

$$Y[\max + 1] = 0 \dots Y[i] \dots Y[0],$$

und wobei diese 3-Operanden-Addition in zwei Abschnitte unterteilt wird.

6. Kryptographie-Verfahren nach Anspruch 5, dadurch gekennzeichnet, daß der erste der beiden Abschnitte so gewählt ist, daß an jeder binären Stelle eine Summe der drei Bits der Operanden  $A$ ,  $B$  und  $C$  gebildet wird, wobei die Summe von  $A[i]$ ,  $B[i]$ , und  $C[i]$  zwischen 0 und 3 liegt, sie also binär mit den zwei Bits  $S1$  und  $S0$  darstellbar ist, und wobei aus den zwei Summenbits in folgender Weise zwei neue Zahlen  $X$  und  $Y$  zusammengestellt werden:

$$\begin{array}{ll} Y[i]: & = \text{niederwertiges Bit von } A[i] + B[i] + C[i], \\ Y[\max + 1]: & = 0, \\ X[i + 1]: & = \text{höherwertiges Bit von } A[i] + B[i] + C[i] \text{ und} \\ X[0]: & = 0. \end{array}$$

( $i = 0, \dots, \max$ ).

7. Kryptographie-Verfahren nach Anspruch 5 und 6, dadurch gekennzeichnet, daß der zweite Abschnitt so gewählt ist, daß die Zahlen  $X$  und  $Y$  in an sich bekannter Weise mit Carry (Übertrag) addiert werden, und daß demgegenüber die Bitaddition ohne Carry des ersten Abschnitts zu einem Zeitpunkt ausgeführt wird, in

dem die normale Additionslogik durch ein Precharge-Signal (Vorbereitungssignal) auf den nächsten Zyklus vorbereitet wird.

8. Kryptographie-Verfahren nach Anspruch 7, dadurch gekennzeichnet, daß die Addition folgende Schritte umfaßt:

- a) Aufteilung der langen bzw. großen Zahlen  $X$  und  $Y$  in kleine Blöcke (32),
- b) Gleichzeitige Berechnung der Carry-Bits innerhalb der Blöcke (32) nach einem an sich bekannten Carry-Look-Ahead-Verfahren, und
- c) Weitergabe des Carry-Bits je eines Blockes zum jeweils nachfolgenden Block, für den Fall, daß sich die Carry-Bits benachbarter Blöcke nicht beeinflussen.

9. Kryptographie-Verfahren nach Anspruch 8, dadurch gekennzeichnet, daß ein durch die Blöcke (32) aktivierbarer Unterbrecher (62) vorgesehen ist, der für den Fall, daß sich Carry-Bits über benachbarte Blöcke hinaus beeinflussen, die erforderliche Zeit für deren Berechnung und Berücksichtigung bereit hält (Fig. 14).

10. Kryptographie-Prozessor zur Durchführung des Kryptographie-Verfahrens, gekennzeichnet durch eine Aneinanderreihung von für die Berechnung der einzelnen Operationen spezialisierten Elementarzellen (10), wobei jeweils mehrere Elementarzellen (10) stufenweise (Fig. 8) zu größeren Blöcken (32; Fig. 9) zusammengefaßt sind, und wobei jedem Block (28; 32) ein baumartiges hierarchisches Carry-Look-Ahead-Element (30) zugeordnet ist, wodurch im Normalfall gewährleistet ist, daß die Zeit zur Addition zweier Zahlen unabhängig von der Länge dieser Zahlen ist, und wobei in jedem Block (32) der Übertrag parallel durchgeführt wird.

11. Kryptographie-Prozessor nach Anspruch 10, dadurch gekennzeichnet, daß die Elementarzellen (10) die folgenden Register und Logik-Bauteile enthalten:

- ein Register (12) für den Multiplikator ( $M$ ),
- ein Code-Register (14),
- ein Datum-Register (16),
- ein UD-Shift-Register (18), in dem während der Berechnung ein Vielfaches des Modulus ( $N$ ) steht, und das außer der Speicherfunktion die Fähigkeit besitzt, den Modulus ( $N$ ) in einem Schritt in eine der beiden Richtungen um mehrere Stellen zu verschieben,
- einen Barrel-Shifter (20), der das Ergebnis der Addition, ein Zwischenergebnis ( $Z$ ) um mehrere Bits verschieben kann,
- einen Bitaddierer (22) ohne Übertragsbit (Carry-Bit), der den ersten Schritt der 3-Operanden-Operation ausführt,
- einen Volladdierer (24), der die beiden im Bitaddierer (22) gewonnenen Zahlen addiert und als ein Zwischenergebnis ( $Z$ ) speichert, und
- ein Carry-Look-Ahead-Element (26), welches ein Übertragsbit berechnet.

12. Kryptographie-Prozessor nach Anspruch 11, dadurch gekennzeichnet, daß alle Komponenten (12—26) parallel arbeiten.

13. Kryptographie-Prozessor nach einem der vorhergehenden Ansprüche 10—12, dadurch gekennzeichnet, daß mehrere Blöcke (28) von Elementarzellen (10) zu größeren Blöcken (32) zusammengefaßt sind, wobei der Übertrag seriell von einem Block zum nächsten weitergegeben wird, und wobei die Carry-Look-Ahead-Elemente (30) der Blöcke (32) wiederum baumartig zusammengestellt werden, und wobei auf jedem übergeordneten Carry-Look-Ahead-Element (30) eines Blockes (32) der Übertrag gleichzeitig berechnet wird, und daß ein sich dabei ergebendes Signal gegebenenfalls einen Unterbrecher (62) ansteuert, wobei der Unterbrecher (62) die von den Carry-Look-Ahead-Elementen (30) kommenden Signale verarbeitet und die Takte für etwa acht Zyklen unterbricht, falls ein Carry-Look-Ahead-Element eines Blockes (32) ein Signal gibt.

14. Kryptographie-Prozessor nach einem der vorhergehenden Ansprüche 10—13, gekennzeichnet durch eine MultMod-Steuereinheit (36; Fig. 12) zur Steuerung der Funktionen der Elementarzellen (10), wobei die MultMod-Steuereinheit (36) folgende Bestandteile umfaßt:

- eine Schiebelogik (50) für die Multiplikation,
- eine Schiebelogik (52) für die Modulo-Operation,
- einen Vergleicher (38), der die obersten Bits des Zwischenergebnisses ( $Z$ ) des Volladdierers (24) mit den obersten Bits von  $1/3$ ,  $1/6$ ,  $1/12$  usw. von Modulus  $N$  parallel miteinander vergleicht,
- einen ersten Begrenzer (54) für die Multiplikation, der den maximalen Schiebebetrags des Zwischenergebnisses ( $Z$ ) im Bedarfsfall begrenzt, und einen zweiten Begrenzer (56) für die Modulo-Operation, der den maximalen Schiebebetrags des UD-Shift-Registers (18) im Bedarfsfall begrenzt, und
- zwei Zähler (58, 60), von denen der erste Zähler (58) die noch zu verarbeitenden Bits des Registers (12) und von denen der zweite Zähler (60) die Position von Modulus  $N$  in einem Puffer (34) angibt.

15. Kryptographie-Prozessor nach einem der vorhergehenden Ansprüche 10—14, gekennzeichnet durch einen als Puffer (34) ausgebildeten Elementarblock mit einer Länge von ca. 20 Bits, der die Look-Ahead-Algorithmen für die Multiplikation und für die Modulo-Operationen voneinander entkoppelt, indem  $m$   $N$  in den Puffer (34) hineinläuft und wobei die MultMod-Steuereinheit (36) durch die Begrenzer (58, 60) gewährleistet, daß der Modulus  $N$  nicht über die Puffergrenze nach oben oder unten hinausläuft.

16. Kryptographie-Prozessor nach Anspruch 10, dadurch gekennzeichnet, daß jeweils zwanzig Elementarzellen (10) zu jeweils einem 20-Zellen-Block (32) zusammengefaßt sind.

17. Kryptographie-Prozessor nach Anspruch 10, gekennzeichnet durch eine Verschlüsselungseinheit (40), einen variabel gestaltbaren Registerblock (46), einer Eingabe/Ausgabeeinheit (44) und einer Hauptsteuer-  
einheit (42), die über Datensammelleitungen miteinander in Wirkverbindung stehen (Fig. 11).

#### Beschreibung

Mit der ständig wachsenden Verbreitung elektronischer Verfahren der Kommunikation und der Informations-Speicherung ist die Forderung nach Geheimhaltung, insbesondere der Geheimhaltung wichtiger Dokumente, wie z. B. Banküberweisungen, Verträge und dergleichen, unverzichtbar geworden.

Während das Problem des Datenschutzes in der Gesetzgebung bereits eine gewisse Berücksichtigung gefunden hat, sind die technischen Probleme zur Durchführung des Datenschutzes mittels der Geheimhaltung von zu übermittelnden Daten bisher noch höchst unbefriedigend gelöst. Die Übertragung von Daten über Funk oder Breitbandkabel erfolgt mehr oder weniger öffentlich. Jedenfalls wird für die Vertraulichkeit der Übertragung keine Garantie übernommen. Die Gefahr des Mißbrauches ist dabei keinesfalls auszuschließen.

Hinsichtlich der Übertragung von Daten über Funk oder Kabel ist diese Gefahr mit technischen Mitteln nicht zu beseitigen. Der Benutzer selbst muß für die erforderliche Sicherheit sorgen. Hierzu gehört auch die Sicherung der Authentizität des Absenders sowie der Manipulationsschutz der Nachricht.

Aus Sicherheitsgründen gilt es deshalb, die zu übertragenden Informationen, Daten, Texte usw. zu verschlüsseln, d. h., derartig umzuwandeln, daß ein Unbefugter sie nicht verstehen kann. Dabei kann allgemein gesagt werden, daß eine Verschlüsselung um so sicherer ist, je komplizierter die der Verschlüsselung zugrunde liegenden Operationen sind.

Bei den als klassisch zu bezeichnenden Verschlüsselungs-Verfahren handelt es sich um symmetrische Methoden, bei denen der Chiffrier- und Dechiffrier-Schlüssel gleichartig, d. h. identisch oder invers sind. Solange der diesbezügliche Schlüssel geheim ist, kann die entsprechend chiffrierte Nachricht öffentlich übertragen werden. Damit aber der Empfänger diese Nachricht verstehen kann, ist es erforderlich, daß dem Empfänger der geheime Kodier-Schlüssel durch einen vertrauenswürdigen Boten zugestellt wird. Diese Art der Zustellung des geheimen Schlüssels ist umständlich und zeitraubend, und zwar besonders dann, wenn mehrere Empfänger mit einer vertraulichen Nachricht versorgt werden sollen. Im übrigen mutet es im elektronischen Zeitalter anachronistisch an, Kuriere für die Übermittlung von geheimen Chiffrier-Schlüsseln einzusetzen.

Demgegenüber stellen die Chiffrier-Methoden nach dem sogenannten Public-Key-Code-Verfahren gedanklich einen großen Fortschritt dar. Diese Public-Key-Code-Verfahren sind durch eine asymmetrische Verschlüsselung gekennzeichnet. Das bedeutet, daß zum Ver- und Entschlüsseln zwei verschiedene Schlüssel benutzt werden. Bei den asymmetrischen Verfahren ist sichergestellt, daß der eine Schlüssel sich nicht ohne Zusatzinformation aus dem anderen berechnen läßt. Einer der beiden Schlüssel kann daher veröffentlicht werden. Aus diesem Grunde haben diese Verfahren die Bezeichnung "Public-Key-Code-Verfahren" erhalten.

Will ein Benutzer der öffentlichen Netze mit anderen Teilnehmern mittels eines Public-Key-Code-Verfahrens Nachrichten austauschen, so muß er ein einziges Mal zwei Schlüssel  $E$  und  $D$  erzeugen. Den Schlüssel  $E$  zum Verschlüsseln macht er über ein Öffentliches Register allen anderen Benutzern zugänglich, den Schlüssel  $D$  zum Entschlüsseln hält er geheim. Darüber hinaus werden bei manchen Verfahren auch die allgemeinen Rechenvorschriften der Verschlüsselung bekanntgegeben, ohne dadurch die Sicherheit der Geheimhaltung des Inhalts der verschlüsselten Nachrichten zu gefährden. Auch ist die Authentizität der Nachricht kein Problem. Die Sicherheit der asymmetrischen Methode beruht darauf, daß es praktisch unmöglich ist,  $D$  aus  $E$  zu berechnen.

Jeder, der einem anderen Benutzer eine Nachricht zusenden möchte, besorgt sich den Schlüssel  $E$  aus dem veröffentlichten Register, verschlüsselt damit die Nachricht und überträgt den so erhaltenen Code im unsicheren (gegebenenfalls digitalen) Netz, beispielsweise dem öffentlichen Telefonnetz. Der adressierte Benutzer (Empfänger) entschlüsselt den empfangenen Code mit seinem geheimen Schlüssel  $D$  und erzeugt so die ursprüngliche Nachricht. Sowohl zur Übermittlung eines Schlüssels als auch zur Übermittlung der Nachricht selbst erübrigt sich somit ein sicherer Übertragungskanal. Der adressierte Benutzer erhält ausschließlich Nachrichten, die mit seinem eigenen Schlüssel verschlüsselt worden sind. Daher braucht er nur auf den eigenen Schlüssel  $D$  zuzugreifen.

Bei diesen Verfahren wird somit eine leichte Verfügbarkeit der Schlüssel erreicht. Ebenso wird der Benutzer der Verwaltung eines umfangreichen persönlichen Schlüsselregisters entoben. Die Schlüsselverwaltung erfolgt nur einmal, und zwar zentral in jedermann zugänglichen Register, z. B. nach Art elektronischer Telefonbücher. Mit dieser Übertragungsprozedur können alle Arten von Übertragungsnetzen (z. B. "ISDN") sicher gemacht werden.

Bei der beschriebenen Ausführungsform des Public-Key-Code-Verfahrens ist noch nicht die Sicherung der Absenderauthentizität sowie der Manipulationsschutz der Nachricht gewährleistet. Prinzipiell ist es aber möglich, fälschungssichere "Unterschriften" in digitaler Form zu übermitteln, und zwar dann, wenn die Reihenfolge der Anwendung der Schlüssel  $E$  und  $D$  vertauschbar ist. Der Absender kann dann eine Signatur erzeugen, die zusammen mit der verschlüsselten Nachricht übertragen wird. Die Signatur ist ein mit dem geheimen Absenderschlüssel  $D$  verschlüsselter "Extrakt" der Nachricht. Zur Überprüfung der Absenderauthentizität erzeugt der Empfänger aus der rekonstruierten Nachricht ebenfalls den Extrakt, entschlüsselt die Signatur mit dem öffentlichen Absenderschlüssel  $E$  und vergleicht beide. Sind sie identisch, so muß die Nachricht vom angegebenen Absender stammen, da nur der Absender den zum Absenderschlüssel  $E$  passenden Schlüssel  $D$  kennt, mit dem die Signatur verschlüsselt wurde.

Mit der Signatur ist die Nachricht auch vor Manipulationen geschützt. Der Absender kann die übertragene

Nachricht nicht abstreiten, da der Empfänger im Besitz einer Signatur dieser Nachricht ist. Andererseits kann der Empfänger die Nachricht nicht verändern, da er für die verfälschte Nachricht keine Signatur erzeugen kann. Diese hängt wegen des Extraktes nicht nur vom Absender, sondern auch von der Nachricht ab. Hierdurch wird ein höherer Schutz als durch die Unterschrift unter einem Dokument gewährleistet.

Das wohl bekannteste Public-Key-Code-Verfahren ist das nach den Anfangsbuchstaben seiner Erfinder Rivest, Shamir und Adleman bekannte RSA-Verfahren. Die Sicherheit dieses RSA-Verfahrens beruht darauf, daß es praktisch unmöglich ist, große Zahlen (z. B. 200 Dezimalstellen) zu faktorisieren, d. h. alle Primzahlen zu finden, durch die diese große Zahl ohne Rest geteilt werden kann.

Das RSA-Verfahren funktioniert folgendermaßen: Zunächst wählt jeder Benutzer des RSA-Systems zwei große Primzahlen  $p$  und  $q$  und eine weitere große Zahl  $E$ . Die Zahlen können z. B. mit dem Zufallszahlen-Generator eines Rechners erzeugt werden. Zur Beantwortung der Frage, ob es sich bei der jeweils vorliegenden Zahl um eine Primzahl handelt oder nicht, stehen Algorithmen zur Verfügung (siehe z. B.: Pomerance, C. "Recent Developments in Primality Testing", Department of Mathematics, University of Georgia, in "The Mathematical Intelligencer" S. 97–104, Vol 3, Nr. 3, 1981).

Das RSA-Verfahren schreibt eine bestimmte Mindestlänge für die Primzahlen nicht vor. Kurze Zahlen machen den Algorithmus schneller, vergrößern aber die Gefahr, daß das Produkt der Primzahlen faktorisiert werden kann. Bei langen Zahlen ist es umgekehrt. 100 Dezimalstellen werden allgemein als ein guter Kompromiß angesehen. Nist das Produkt aus den Primzahlen  $p$  und  $q$ . Das Zahlenpaar  $(E, N)$  ist der öffentliche Schlüssel, während die Primzahlen  $p$  und  $q$  ausschließlich dem Empfänger einer Nachricht bekannt sind.

Zur Chiffrierung der Nachricht verwandelt der Absender zunächst seinen Text in eine Kette von Dezimalzahlen. Diese Kette wird dann in gleichlange Glieder  $P+i < N$  zerlegt. Diese Glieder werden sodann einzeln chiffriert, indem man sie jeweils in die  $E$ -te Potenz erhebt und dann modulo  $N$  bildet, d. h., es entstehen die Zahlen  $C_i = P_i^E \text{ modulo } N$ . Diese Zahlen  $C_i$  werden dann über einen unsicheren Kanal geschickt. Zur Auswertung der Zahlen ist es erforderlich, ihren Exponenten modulo  $\Phi(N)$  zu berechnen, wobei  $\Phi(N) = (p-1) \cdot (q-1)$  ist. Da ausschließlich der Empfänger die Primzahlen  $p$  und  $q$  kennt, kann nur er den Dechiffrier-Schlüssel  $D = E^{-1} \text{ modulo } \Phi(N)$  berechnen. Zu diesem Zweck erhebt der Empfänger jede empfangene Zahl  $C_i$  in die  $D$ -te Potenz und reduziert modulo  $N$ . Da  $C_i \text{ modulo } N = P_i^E \text{ modulo } N$  und  $ED \text{ modulo } \Phi(N) = 1$ , ergibt die Operation  $P_i^{ED} \text{ modulo } N$  wieder die Zahlenblöcke des Klartextes.

Außer den als "klassisch" bezeichneten Chiffrier-Verfahren zeigen auch die bisher bekanntgewordenen Public-Key-Code-Verfahren gravierende Mängel. Sowohl software- als auch hardwaremäßige Realisierungen scheiterten bisher stets an dem immensen Aufwand und den damit verbundenen hohen Kosten.

Nach dem bishereigen Stand der Chip-Entwicklung ist es nicht möglich, einen Universal-Rechner softwaremäßig mit dem RSA-Algorithmus (bei Zugrundelegung von 200 Dezimalstellen) so zu programmieren, daß akzeptable Verschlüsselungsgeschwindigkeiten bzw. Verschlüsselungsraten erreicht werden.

Auch kann die RSA-Funktion (Potenzierung mit anschließender Modulo-Operation) nicht direkt in ein VLSI-Layout (VLSI = Very Large Scale Integration) umgesetzt werden, weil es keine direkten Potenzierschaltungen gibt. Die geschilderte Erkenntnis hat bereits seit mehreren Jahren zu dem Wunsch nach speziellen Hardware-Lösungen geführt, um die Potenzierung so in Einzelschritte zu zerlegen, daß eine hinreichende Verschlüsselungsgeschwindigkeit bzw.-Rate möglich ist.

Die heute bekannten Realisierungen der Public-Key-Code-Verfahren benötigen aber sehr viel Rechenzeit. Software-Lösungen haben Ver- bzw. Entschlüsselungsraten von 10 bis 20 Bit/sec. Auch erste bekannte Hardware-Lösungen erreichen nicht mehr als 1 200 Bit/sec. Die einzige bisher realisierte Ein-Chip-Lösung stammt von Rivest (Rivest, R. L., "A Description of a Single-Chip Implementation of the RSA Cipher", Laboratory for Computer Science, MIT, Cambridge, Massachusetts, in "LAMBDA Magazine 1", S. 14–18, Nr. 3, 1980). Bei diesem Vorschlag wurde eine relativ einfache Entwurfsmethode gewählt, die insbesondere darin besteht, daß mit einer herkömmlichen arithmetisch-logischen Elementarzelle eine 512 Bit breite arithmetisch-logische Einheit (ALU) konstruiert wurde. Diese arithmetisch-logische Einheit ist so aufgebaut, daß mit ihr sehr verschiedene Operationen ausgeführt werden können. Die in Kauf genommene Redundanz schlägt sich in einer Verschlüsselungsrate von 1 200 Bit/sec. nieder. Dabei ist eine 4  $\mu$ m-NMOS-Technologie benutzt worden (Extrapoliert auf eine 2  $\mu$ m-CMOS-Technologie würde sich bei einer Schlüssel-Länge von 660 Bits eine Verschlüsselungsrate im Bereich von 2 500 Bit/sec. ergeben).

Diese Lösung ist in der Praxis allerdings nicht akzeptabel, da die Schnittstellen der digitalen Netze mit sehr viel höheren Datenraten arbeiten, z. B. arbeiten die ISDN-Schnittstellen mit 64 Bit/sec.

Es ist auch schon ein Kryptographie-Prozessor vorgeschlagen worden, der aus zwei Chips besteht, und mit deren Hilfe 336 Bit lange Zahlen des RSA-Algorithmus aufgearbeitet werden können (Rieden, R. F., J. B. Snyder, R. J. Widman and W. J. Barnard, "A Two-Chip Implementation of the RSA Public-Key Encryption Algorithm", Digest of Papers for the 1982 Government Microcircuit Applications Conference (November 1982), 24–27).

Der schnellste bekannte RSA-Prozessor ist mit dem Vorschlag von NEC/Miyaguchi gegeben (Miyaguchi, S., "Fast Encryption Algorithm for the RSA Cryptography System", Proceedings COMPCON 82). Er arbeitet pro Zyklus 8 Bits des Multiplikators ab und erreicht dabei eine Geschwindigkeit von 29 000 Bits/sec. Da er aber für die praktische Ausführung die hohe Anzahl von 333 Chips benötigt, ist er in wirtschaftlicher Hinsicht natürlich völlig indiskutabel.

Ein genereller Nachteil der Mehr-Chip-Implementierungen besteht nicht nur in den proportional mit der Anzahl der erforderlichen Chips steigenden Hardware-Kosten, sondern vor allem in der fehlenden Gewähr für Sicherheit. Wenn die Signale, die von einem zum anderen Chip übertragen werden, zugänglich sind, so kann anhand der übertragenen Signale der Geheimcode gebrochen werden. Deshalb ist es aus Gründen der Kryptographie-Sicherheit wesentlich, daß alle Kryptographie-Algorithmen möglichst von einem einzigen Chip durch einen Tresor geschützt werden könnten.

In der DE-PS 32 28 018 ist ein Schlüsselssystem für RSA-Kryptographie beschrieben, welches ebenfalls einen extrem hohen Hardware-Bedarf erfordert. Im Vergleich mit dem ursprünglichen RSA-Algorithmus soll bei dem bekannten Schlüsselssystem die Verschlüsselungsrate um den Faktor 4 erhöht werden, was aber nur eine bescheidene Verbesserung darstellt. Zu diesem Zweck wird eine feste Anzahl von vier Bits gleichzeitig verarbeitet, wofür mehrere Multiplikatoren erforderlich sind, und wofür insgesamt 14 Addierer angegeben werden.

Theoretisch ist es zwar denkbar, daß der Prozessor für das bekannte Schlüsselssystem gemäß der DE-PS 32 28 018 viermal so schnell wie die direkte Verwendung des ursprünglichen RSA-Algorithmus ist, da aber die Signalwege sehr viel größer sind als bei einer Abtastung von jeweils einem einzigen Bit, ist in der Praxis kaum ein effektiver Zeitgewinn zu erwarten.

Auch ein gedanklich angenommener Universal-Chip mit einer 100-fachen Rechnerdichte wäre übrigens nicht in der Lage, den bekannten RSA-Algorithmus abzuarbeiten. Aus diesem Grunde käme, wenn überhaupt, nur ein Spezial-Kryptographie-Chip in Frage, allerdings würden sich dabei erhebliche Kühlprobleme einstellen, denn bei derartig hochspezialisierten Chips wären — im Unterschied zu Universalchips — sämtliche Transistorfunktionen fast ständig im Einsatz. Dies ist mit beträchtlichen Verlustleistungen verbunden, die wegen der angenommenen 100-fachen Rechnerdichte aus einer 100-fachen (Verlust-) Leistungsdichte zur Folge haben würde.

Daß die damit verbundenen Kühlprobleme nicht unbeträchtlich sind, zeigt der aufwendige Vorschlag, die Kühlung mittels verflüssigter Edelgase durchzuführen, die durch im Silizium-Chip führende Bohrungen hindurchfließen. Andererseits ist zu berücksichtigen, daß sich als Folge einer ungenügenden Kühlung eine erheblich verkürzte Lebensdauer der Chips und eine erhöhte Fehlerquote bei den Verschlüsselungsoperationen ergibt. Außerdem würde der angenommene Universal-Chip mit der hohen Rechnerdichte von den räumlichen Abmessungen her so groß ausfallen, daß eine praktikable Anwendung außer Betracht bleiben müßte.

Hier greift nun die Erfindung ein, der die Aufgabe zugrunde liegt, ein Kryptographie-Verfahren anzugeben, welches bei hinreichender Sicherheit eine so schnelle Rechengeschwindigkeit ermöglicht, daß eine kommerzielle praktikable Anwendung des an sich bekannten RSA-Verfahrens möglich ist. Außerdem soll durch die Erfindung ein Kryptographie-Prozessor zur Durchführung des Verfahrens geschaffen werden, der die gestellten Anforderungen bei kleiner handlicher Bauweise bzw. Chip-Abmessungen ermöglicht.

Dieses Ziel erreicht die Erfindung verfahrensmäßig bei dem im Oberbegriff des Anspruchs 1 genannten Kryptographie-Verfahren durch die im kennzeichnenden Teil des Anspruchs 1 genannten Merkmale, wobei den Anforderungen einer digitalen Schnittstelle eines ISDN-Netzes genügt wird.

Ein wesentlicher Gesichtspunkt der Erfindung ist die neuartige Anwendung eines Look-Ahead-Algorithmus für die Division. Durch diesen Schritt wird es nämlich möglich, ein Look-Ahead-Verfahren auch bei der Multiplikation anzuwenden. Somit sind beim Abarbeiten mehrerer Bits nur einfache Additionen und Subtraktionen erforderlich, d. h. ein zusätzliches Multiplizieren kann entfallen.

Die Vorteile des neuen Verfahrens basieren also im einzelnen darauf, daß der gesamte Kryptographie-Algorithmus sukzessive so weit in kleinere Schritte zerlegt wird, bis jeder Rechenschritt in einfacher Weise direkt mit einer Hardware-Auslegung korrespondiert.

Vorteilhafte Weiterbildungen und zweckmäßige Ausgestaltungen der Erfindung sind in den Unteransprüchen angegeben.

Jeweils bei der Gegenüberstellung der einzelnen Operationen zeigen sich bereits die Vorteile der erfindungsgemäßen Anordnung gegenüber dem Stand der Technik. Durch die Umwandlung des Exponenten-Algorithmus in eine Folge von Multiplikationen, wobei nach jeder einzelnen Multiplikation eine Modulo-Operation ausgeführt wird, wird verhindert, daß die Zwischenergebnisse nicht, wie bei der Potenzierung üblich, ins Astronomische anwachsen ( $D$  und  $E$  haben je 200 Dezimalstellen).

Dadurch, daß darüberhinaus die Multiplikation in Einzelschritte zerlegt wird, bei der die Multiplikation in eine Folge von Additionen umgewandelt wird, kann die Berechnung schneller erfolgen. So wird zur Realisierung außerdem weniger Fläche auf dem Chip benötigt.

Die weiter unten erläuterte Reduktion der Modulo-Operation in eine Folge von Subtraktionen wird mit derselben Additionslogik berechnet, denn eine Subtraktion kann als Addition mit umgekehrten Vorzeichen behandelt werden.

Da die Multiplikation im Ring über  $N$  ausgeführt wird, kann bereits nach jeder Addition eine Modulo-Operation ausgeführt werden. Auch hierdurch werden große Zahlen vermieden und beträchtliche Rechenzeit eingespart. Sämtliche Zahlen jedes Schrittes sind nun kleiner als  $N$ . Auf diese Weise wird die maximal erforderliche Größe des Speichers auf die Länge von  $N$  reduziert, was mit einer Halbierung der erforderlichen Chipfläche verbunden ist.

Beträchtliche Bedeutung für das Kryptographie-Verfahren liegt in der vorteilhaften Anwendung von Look-Ahead-Algorithmen, denn hierdurch wird die maximale Anzahl erforderlich werdender Additionen für die Multiplikationen und Subtraktionen für die Modulo-Operationen weiter beträchtlich reduziert. Da sich die Einführung der Look-Ahead-Algorithmen in einem Gewinn an Rechengeschwindigkeit auszahlt, ergibt sich aber erst durch die Einführung der erfindungsgemäßen neuen Look-Ahead-Algorithmus für die Modulo-Operation, denn die bloße Anwendung bekannter Look-Ahead-Algorithmen auf die Multiplikationen würde keinesfalls einen Zeitgewinn erbringen. Erst wenn die mittlere Reduktion des mit der Modulo-Operation verbundenen Rechenaufwands der mit der Multiplikation möglichen Rechenreduktion entspricht, ergibt sich ein optimaler Algorithmus, der die Rechenzeit auf etwa  $1/3$  reduziert. Diese vorteilhafte Einsparung an Rechenzeit hängt mit dem weiter unten noch erläuterten "Schwimmen" zusammen; während  $Z$  absolut verschoben wird, erfolgt die Verschiebung von  $N$  relativ zu  $Z$ , so daß die beiden Verschiebe-Raten voneinander entkoppelt sind.

Der letzte Vorteil in der Kette von Verfahrensschritten besteht in der Verknüpfung der sich aus der Multiplikation ergebenden Addition und der sich aus der Modulo-Operation ergebenden Subtraktion zu einer einzigen Operation, der 3-Operanden-Addition. Mit ihrer Hilfe braucht die Zykluszeit nicht erweitert zu werden, denn für

die 3-Operanden-Addition wird die gleiche Zeit benötigt wie für eine einfache Addition. Hierdurch wird eine Verdoppelung der Rechengeschwindigkeit bewirkt.

Einerseits resultiert ein Zeitgewinn aus den vorteilhaften Anwendungen von mathematischen Transformationen auf die einzelnen Verfahrensschritte, andererseits ergeben sich aus der erfindungsgemäßen Architektur des Kryptographie-Prozessors weitere Verbesserungen. Durch die Organisation zu baumartigen Strukturen können die einzelnen Elemente gleichzeitig eine größere Anzahl von Informationen abarbeiten. Auch hierdurch wird die Rechenzeit weiter beträchtlich verkürzt.

Mit der erfindungsgemäßen Blockstruktur reduziert sich die Rechenzeit für eine 660-Bit-Addition auf die Länge einer 20-Bit-Addition.

Zusammenfassend ist festzustellen, daß der erfindungsgemäße Kryptographie-Prozessor die Ver- und Entschlüsselung mit einer Rate von 64 000 Bits pro Sekunde durchführt. Dies gilt auch für den ungünstigsten Fall, wo der Schlüssel die maximale Länge von 660 Bits aufweisen sollte.

Die mit dem erfindungsgemäßen Kryptographie-Verfahren bzw. mit dem entsprechenden Kryptographie-Prozessor erzielbaren Vorteile werden augenscheinlich, wenn man seine "Über-alles-Effizienz" mit der einer Software-Implementierung, z. B. auf einem 16 Bit breiten Bit-Slice-Prozessor (BSP) vergleicht, der speziell auf die Verschlüsselungsaufgabe zugeschnitten worden ist: Heute erhältliche BSP's haben eine Taktfrequenz von ca. 10 MHz. Sie können in einem Zyklus zwei 16-Bit-Worte addieren bzw. subtrahieren und gleichzeitig das Ergebnis um 1 Bit verschieben. Einen Barrel-Shifter haben sie nicht, so daß eine Shift-Operation seriell ausgeführt werden muß. Unter diesen Bedingungen hat ein Look-Ahead negative zeitliche Auswirkungen. Für die Zahl  $A$  der Zyklen zur Ausführung einer Operation wird angenommen, daß der BSP für jeden Schritt seiner Hauptschleife nur einen Zyklus benötigt. Sie besteht aus (vgl. Fig. 3):

1.  $Z[i] := Z[i] + P[i]$  und
2.  $Z[i] := Z[i] + -N[i]$ , verschiebe  $Z[i]$  um 1 Bit.

Das Mikroprogramm der BSP's kann so ausgelegt werden, daß sich die Schleife auf den zweiten Schritt reduziert, wenn der erste aufgrund des Tests des entsprechenden Bits im Multiplikator entfallen kann. Da der erste Schritt mit einer Wahrscheinlichkeit von 1/2 ausgeführt werden muß, hat  $A$  einen Wert von 1,5. Für den Vergleich folgt:

$$\frac{V_{RSA, KP}}{V_{RSA, allz.}} = \frac{f_{KP}}{f_{BSP}} \cdot \frac{Erw(sz) \cdot L(N) \cdot A}{B} = \frac{30 \text{ MHz}}{10 \text{ MHz}} \cdot \frac{2,27 \cdot 660 \cdot 1,5}{16} \approx 421$$

Der Vergleich zeigt, daß der erfindungsgemäße Kryptographie-Prozessor selbst spezialisierten Abstimmungen aus Hard- und Software um mehr als zwei Größenordnungen überlegen ist. Ist der RSA-Algorithmus auf normalen Rechenanlagen nur softwaremäßig implementiert, kann von Unterschieden im Bereich von 10 000 ausgegangen werden, da die Hauptschleife bei weitem nicht so effizient ausgeführt werden kann.

Verglichen mit dem eingangs erwähnten Rivest-Prozessor ergibt sich eine 50-fache Einsparung an Verschlüsselungszeit trotz einer um ca. 30% größeren Schlüssellänge. Diese gewaltige Steigerung der Über-Alles-Effizienz wurde sowohl mit der erfindungsgemäßen Anwendung neuer Verfahrensschritte als auch durch eine spezielle Chip-Architektur ermöglicht.

Der erfindungsgemäße Kryptographie-Prozessor arbeitet als sogenannter Coprozessor. Er besitzt zwei DMA-Kanäle für das Daten I/O und einen 8 Bit breiten Datenbus. Die Kodier- und die I/O-Einheit arbeiten parallel. Er stellt eine "Kryptographie-Box" dar, bei der die Ver- und Entschlüsselungen von außen nicht beeinflußt werden können, bei der eine Signatur erzeugt wird, und zu der die Schlüssel nur im verschlüsselten Zustand übertragen zu werden brauchen.

Im folgenden wird das erfindungsgemäße Kryptographie-Verfahren und der darauf aufbauende Kryptographie-Prozessor, mit deren Hilfe Daten nach der Public-Key-Code-Methode von Rivest, Shamir und Adleman (RSA) ver- und entschlüsselt werden, anhand eines Ausführungsbeispiels näher beschrieben.

Im ersten Teil der Beschreibung, der sich mit dem erfindungsgemäßen Kryptographie-Verfahren befaßt, werden die gegenüber dem ursprünglichen RSA-Verfahren gegebene Modifizierungen erläutert. Deshalb wird dabei häufig auf den ursprünglichen RSA-Algorithmus Bezug genommen. Da das weitere Ziel der Erfindung in der Realisierung des Verfahrens in einem effizienten VLSI-Layout besteht, bleibt es nicht aus, bereits bei der Beschreibung des Verfahrens auf die entsprechenden Hardware-Möglichkeiten hinzuweisen. Insbesondere werden komplexe Operationen soweit in Grundoperationen (Addieren, Subtrahieren, Verschieben usw.) zerlegt, bis sich jeder Schritt unmittelbar in den später folgenden VLSI-Entwurf umsetzen läßt. Deshalb spielt die Betrachtung des Algorithmus aus der Sicht der Hardware eine wichtige Rolle. Viele Schrittfolgen des RSA-Algorithmus werden durch vorteilhaftere ersetzt.

Da die Entschlüsselung mathematisch identisch mit der Verschlüsselung ist, wird deshalb im folgenden auf die beiden Vorgänge nicht gesondert eingegangen.

In der Zeichnung zeigen:

Fig. 1a ein Flußdiagramm eines Algorithmus für die Potenzierung für die Ent- oder Verschlüsselung eines Datums nach dem ursprünglichen RSA-Verfahren,

Fig. 1b ein Flußdiagramm eines Algorithmus für die Potenzierung für die Ent- und Verschlüsselung des gleichen Datums gemäß Fig. 1a nach dem erfindungsgemäßen Verfahren,

Fig. 2 ein Flußdiagramm des seriellen Algorithmus für die in Fig. 1 erforderliche Multiplikation, wobei die Multiplikanten Elemente der natürlichen Zahlen sind,

Fig. 3a ein Flußdiagramm des Multiplikations-Algorithmus mit einem zusätzlichen Modulschritt, wodurch die

Multiplikatanten hier Elemente eines Restklassenringes über  $N$  sind,

Fig. 3b ein Flußdiagramm gemäß Fig. 3a, wobei die Modul-Rechnung auf eine Subtraktion reduziert ist,

Fig. 4 ein Flußdiagramm eines Look-Ahead-Algorithmus für die Multiplikation, der die Look-Ahead-Parameter seriell berechnet,

Fig. 5 ein Flußdiagramm eines Look-Ahead-Algorithmus für die Modulo-Operation, der die Look-Ahead-Parameter seriell berechnet,

Fig. 6a ein Flußdiagramm gemäß Fig. 1b, wobei die Multiplikation und anschließende Modulo-Operation zu einem MultMod-Schritt zusammengefaßt sind,

Fig. 6b ein Flußdiagramm des in Fig. 6a verwendeten erfindungsgemäßen MultMod-Verfahrens, ausgeführt mit Look-Ahead,

Fig. 7 eine Elementarzelle zur Realisierung einer MultMod-Schleife in einem Schritt,

Fig. 8 die Zusammenfassung von vier Elementarzellen gemäß Fig. 7 zu einem 4-Zellen-Block mit einem hierarchischen Carry-Look-Ahead (CLA)-Element,

Fig. 9 die hierarchische Zusammenfassung von jeweils fünf 4-Zellen-Blöcken gemäß Fig. 8 zu einem 20-Zellen-Block

Fig. 10 eine vollständige Verschlüsselungseinheit mit mehreren 20-Zellen-Blöcken gemäß Fig. 9, sowie mit einer Steuereinheit,

Fig. 11 das Blockschaltbild eines Kryptographie-Prozessors,

Fig. 12 ein Blockschaltbild einer Steuereinheit gemäß Fig. 10 nach Vorgabe der Look-Ahead-Algorithmen gemäß Fig. 4 und 5,

Fig. 13 ein hierarchisches Carry-Look-Ahead-Element, wie es bei den 4-Zellen-Blöcken gemäß Fig. 8 zur Anwendung gelangt,

Fig. 14 die Verschaltung der Carry-Look-Ahead-Elemente gemäß Fig. 13 innerhalb der Hierarchie der 20-Zellen-Blöcke,

Fig. 15 ein Zustandsdiagramm bezüglich einiger Schrittfolgen des in Fig. 10 als Puffer ausgebildeten obersten 20-Zellen-Blockes,

Fig. 16 eine schematische Blockstruktur zur Verdeutlichung des Informationsflusses, und

Fig. 17 einen Floorplan der Anordnung von Elementarzellen auf einem Chip.

Unter Bezugnahme auf Fig. 1 wird nachfolgend zunächst der Verfahrensschritt für die Potenzierung erläutert.

Bei der Zerlegung in einfache Grundoperationen wird die Potenzierung in durchschnittlich  $1,5 \cdot L(E)$  Multiplikationen zerlegt.  $E$  ist der Exponent und  $L(x)$  ist definiert als

$L(x) := \text{Anzahl der binären Ziffern von } x$

Die Zeitkomplexität des Algorithmus ist  $O(L(E))$ . Seine Grundidee ist, den Exponenten binär darzustellen, also in eine Summe von Zweierpotenzen zu verwandeln. Mit den Potenzgesetzen wird die Summe im Exponenten in ein Produkt von Potenzen der zu potenzierenden Zahl  $P$  umgeformt. Die  $e$ -te Potenz hat als Exponent die  $e$ -te Zweierpotenz oder die Null, je nachdem, ob an der  $e$ -ten Stelle im ursprünglichen Exponenten eine 1 oder eine 0 steht. Die Faktoren sind also Quadrate bzw. die Zahl 1.

$$P^E = P^{\sum_{e=0}^{L(E)-1} E_e \cdot 2^e} = \prod_{e=0}^{L(E)-1} P^{E_e \cdot 2^e}; E_e \in \{0,1\}$$

$$P^{2^e} = P^{2 \cdot 2^{e-1}} = (P^{2^{e-1}})^2$$

Die Stelle des niederwertigsten Bits wird definitionsgemäß mit Null bezeichnet. Deshalb steht das höchstwertigste Bit an der Stelle  $L(E) - 1$ .

Das  $(e + 1)$ -te Quadrat läßt sich leicht durch Quadratur des  $e$ -ten berechnen. Es ist deshalb vorteilhaft, für das Produkt ein eigenes Register  $C$  zu reservieren. Der Inhalt von Register  $P$  wird dann in jedem Schritt quadriert und wieder darin gespeichert. Nach der Quadratur enthält  $P$  das  $e$ -te Quadrat, da nach dem  $(e - 1)$ -ten Schritt in  $P$  das  $(e - 1)$ -te Quadrat eingetragen worden war.

Im Zwischenregister  $C$  steht zu Beginn die 1. Steht im Exponenten  $E$  an  $e$ -ter Stelle eine 1, so wird im  $e$ -ten Schritt  $C$  mit  $P$  multipliziert und wieder darin gespeichert, andernfalls wird  $C$  nicht verändert. Da das Register  $P$  zu diesem Zeitpunkt das  $e$ -te Quadrat enthält, wird das obige Produkt, wegen der Gleichheit,  $P$  hoch  $E$  berechnet. Nach dem letzten Schritt steht das Ergebnis im Register  $C$ .

Das im Flußdiagramm der Fig. 1a dargestellte RSA-Verfahren enthält im oberen Teil den Potenzialgorithmus. Im unteren Teil der Fig. 1a wird im letzten Schritt  $C \bmod N$  berechnet, d. h. der letzte Schritt des RAS-Algorithmus. Weil auf die Restklassenarithmetik während der Potenzierung verzichtet wurde, hat  $C$  bei großen Zahlen eine astronomische Stellenzahl angenommen.

Dies verhindert der in Fig. 1b dargestellte Algorithmus des erfindungsgemäßen Verfahrens. Er nutzt das Kongruenzgesetz

$$(a \bmod c) \cdot (b \bmod c) = (a \cdot b) \bmod c$$

Das jeweils entstehende Produkt wird durch die Modulorechnung auf den Repräsentanten der Restklasse abgebildet. Der Repräsentant ist das Element der Restklasse, das auch Element des Ringes ist. Dieses Element ist eindeutig, d. h. es gibt nur ein Element in jeder Restklasse, das die Bedingung erfüllt.

Bezogen auf den Algorithmus bedeuten  
 $a, b$  Produkte aus dem Schritt  $e - 1$ , und  
 $c$  der Modul.

Die Aussage der Kongruenz ist: das Ergebnis der Rechnung im Schritt  $e$  fällt in dieselbe Restklasse,

- a) wenn man die entstandenen Produkte aus dem Schritt  $e - 1$  auf ihre Repräsentanten abbildet und dann im Schritt  $e$  die Repräsentanten miteinander multipliziert oder
- b) wenn man im Schritt  $e$  die Produkte aus dem Schritt  $e - 1$  miteinander multipliziert und dann dieses Produkt auf seinen Repräsentanten abbildet.

Der Fall a wird im rechten Algorithmus (Fig. 1b) bei jedem Schleifendurchlauf auf die Produkte angewandt. Der Fall b ist im linken Algorithmus (Fig. 1a) realisiert, allerdings nur ein einziges Mal als letzter Schritt des Algorithmus. Durch die ständige Abbildung haben die benutzten Register im rechten Algorithmus eine planbare Größe bekommen. Die Zahlen, die sie speichern müssen, sind maximal doppelt so lang wie die Länge des Modulus. Dies ist der Fall in dem Zeitraum zwischen Multiplikation und der Modulorechnung.

Die einzelnen Schritte des umgeformten Algorithmus lassen sich auf der Ebene der Potenzierung nicht weiter zerlegen. Es besteht auch nicht mehr die Notwendigkeit dafür. Die Chip-Fläche, die die Potenzierung benötigt, hat durch die ständige Modulorechnung eine obere Grenze erhalten.

Nachfolgend wird nun unter Bezugnahme auf Fig. 2 der Verfahrensschritt für die Multiplikation erläutert. Bei dem erfindungsgemäßen Verfahren wird die Multiplikation durch einen seriellen Algorithmus gelöst. Er zerlegt die Multiplikation in  $L(M)$  Shift-Operationen und durchschnittlich  $0,5 \cdot L(M)$  Additionen. Mit "M" ist im folgenden der Multiplikator bezeichnet.

Der Platzbedarf hängt linear von  $L(M)$  ab, denn für diesen Algorithmus ist eine Arithmetik-Lodic-Unit (ALU) der Größe  $L(M)$  vorgesehen, so daß die Addition in einem Schritt geschieht. Das gleiche gilt dann auch für die Shift-Operation. Beide benötigen daher eine konstante Zeit für ihre Operation. Zudem kann die Addition parallel zur Shift-Operation ausgeführt werden. Daraus folgt für die Zeitkomplexität

$$T_{Mul} = L(M) * \max(T_{Shift}, T_{Add}) = c * L(M).$$

Sie hängt, wie der Platzbedarf, linear von der Länge von  $M$  ab. Wenn der benötigte Platz allerdings die Möglichkeiten der Integration überschreitet, d. h.  $M$  und damit  $L(M)$  einen bestimmten Wert übersteigt, ist dieser Algorithmus nur in modifizierter Form verwendbar. Da dies bei der ins Auge gefaßten Größe von 660 Binärstellen (200 Dezimalstellen) nicht der Fall ist, wird dieses Problem in diesem Ausführungsbeispiel nicht diskutiert.

Der in Fig. 2 als Flußdiagramm dargestellte serielle Algorithmus für die Multiplikation baut, ähnlich wie der vorher anhand der Fig. 1 beschriebene Algorithmus für die Potenzierung, auf der Binärdarstellung eines Eingabeparameters auf. Hier ist es der Multiplikator  $M$ :

$$P * M = P * \sum_{m=0}^{L(M)-1} M_m * 2^m = \sum_{m=0}^{L(M)-1} P * M_{L(M)-m} * 2^{L(M)-m}.$$

Die Multiplikation wird in Additionen zerlegt.  $P$  wird im Schritt  $m$  zum Zwischenergebnis  $Z$  addiert, wenn an der  $(L(M) - m)$ -ten Stelle im Multiplikator eine 1 steht, andernfalls bleibt  $Z$  unverändert. Danach wird die Schleife noch  $(L(M) - m)$ -mal ausgeführt. Wegen der Verdoppelung von  $Z$  zu Beginn jedes Schleifendurchlaufs wird die Summe  $Z + P$  des  $m$ -ten Schrittes  $(L(M) - m)$  mal verdoppelt. Das entspricht der Multiplikation mit der Zweierpotenz.

Zusammenfassend nutzt der Algorithmus aus, daß eine Multiplikation mit einer Binärziffer entweder den Multiplikanten selbst oder Null ergibt. Weiterhin führt er die in jedem Schritt erforderliche Multiplikation mit einer Zweierpotenz auf eine Verdoppelung von  $Z$  zurück. In der Binärdarstellung ist die Verdoppelung eine einfache Shift-Operation um ein Bit nach links (definitionsgemäß steht das niederwertigste Bit rechts).

Der Modulo-Verfahrensschritt ist schematisch in Fig. 3 dargestellt. Während der Potenzierung ist nach jeder Multiplikation eine Modulo-Operation auszuführen, um eine zum Produkt kongruente Zahl aus dem Restklassenring zu erhalten. Der in Fig. 2 beschriebene Algorithmus betrachtet die beiden Multiplikanten als Elemente der natürlichen Zahlen, nicht des Restklassenringes über  $N$ . Deshalb wird im Potenzialgorithmus nach jeder Multiplikation ein Moduloschritt ausgeführt.

Nach dem erfindungsgemäßen Verfahren wird auch die Multiplikation in diesem Restklassenring ausgeführt. Dafür wurde der herkömmliche Algorithmus an einer Stelle verändert: Am Ende der Schleife wird das Zwischenergebnis  $Z$  auf seinen Repräsentanten abgebildet.

Das ist notwendig, weil  $Z$  erstens verdoppelt und zweitens  $P$  (im ungünstigsten Fall) zu ihm addiert wurde. Deshalb kann  $Z$  am Ende der Schleife Werte haben, die größer oder gleich des Modulus  $N$  sind.

Wird dagegen zum Schluß noch ein Moduloschritt hinzugefügt, hat  $Z$  nach dem Verlassen der Schleife immer Werte, die im erlaubten Zahlenbereich des Ringes liegen. Das Kongruenzgesetz, das es erlaubt, den Moduloschritt vom Potenzialgorithmus in den Multiplikationsalgorithmus zu verlegen, lautet

$$(a \bmod c) + (b \bmod c) = (a + b) \bmod c.$$

Wie bei der Potenzierung ist auch hier die Aussage der Kongruenz: das Ergebnis der Rechnung im Schritt  $m$

fällt in dieselbe Restklasse,

a) wenn man die entstandene Summe im Schritt  $m - 1$  auf ihren Repräsentanten abbildet und dann im Schritt  $m$  mit diesem weiterrechnet oder

b) wenn man im Schritt  $m$  zu der Summe aus dem Schritt  $m - 1$  etwas addiert und dann diese Summe auf ihren Repräsentanten abbildet.

Da die Multiplikation in eine Summenfolge umgewandelt wurde, lautet der Schluß: Es ergibt das gleiche Ergebnis, zwei Zahlen zu multiplizieren und dann die Modulorechnung auszuführen, oder nach jeder Addition in der zerlegten Multiplikation sofort Modulo zu rechnen.

Das Zwischenergebnis  $Z$  des in Fig. 3a dargestellten Flußdiagramms kann in der Schleife nicht beliebig große Werte annehmen, wenn es beim Schleifeneintritt einen kleineren Wert als  $N$  hatte

$$N > Z, P = > 3 \cdot N > Z : = 2 * Z + P.$$

In dem erfindungsgemäßen Verfahren ist die herkömmliche Modulorechnung durch eine einzige Subtraktion ersetzt.

Wenn  $Z$  am Ende der Schleife größer oder gleich  $N$  ist, wird lediglich  $N$  bzw.  $2N$  von  $Z$  subtrahiert, und der Wert von  $Z$  ist wieder kleiner als  $N$ . Diese Schritte sind im Flußdiagramm der Fig. 3b enthalten.

Für die Subtraktion wird keine zusätzlich Logik benötigt, denn nach der Negation des Subtrahenten wird sie zu einer Addition und ist mit der Additionslogik berechenbar:

$$a + b = a + (-b)$$

Eine Zahl wird negiert, indem jedes einzelne Bit negiert wird. Dazu muß abschließend noch die Zahl 1 addiert werden. Das ist im VLSI-Entwurf mit einem Inverter pro Bit realisierbar. Da jedoch in der Speicherzelle beide Informationen vorliegen, das Bit und das invertierte Bit, wird auf einen zusätzlichen Inverter verzichtet.

Bei der Addition zweier Zahlen wird an die niederwertigsten Bits kein Übertragsbit (Carrybit) übergeben. Soll nun subtrahiert werden, werden die negierten Speicherbits an die Additionslogik angelegt und gleichzeitig wird den niederwertigsten Bits ein Carrybit signalisiert.

Dieser Verfahrensschritt, der die Multiplikation mit der Modulo-Operation in erfindungsgemäßer Weise miteinander verbindet, wird im folgenden MultMod genannt.

Die Erhöhung der Rechengeschwindigkeit durch Look-Ahead-Verfahren läßt sich anhand von Fig. 4 erläutern. Analysiert man den MultMod-Algorithmus und bedenkt die Möglichkeiten der Parallelisierung, so ergibt sich, daß viele Schritte umsonst ausgeführt werden. Genauer gesagt, ganze Schleifendurchläufe (Zyklen) können entfallen, wenn außer den beiden ersten, unabwendbaren Schritten keine der bedingten Schritte auszuführen sind.

Entfällt ein Zyklus, dann wird kein Schritt der Schleife ausgeführt, auch die unbedingten nicht. Dies muß bei dem nächsten, nicht ausgefallenen Zyklus bedacht werden. Zuerst muß jedoch berechnet werden, wieviel Zyklen übersprungen werden können. Sei nun  $sz - 1$  die Anzahl der übersprungenen Zyklen ("sz" ist der Schiebebetrags (Shift-Betrags) der Multiplikation und behält im folgenden diese Bedeutung). Mit dieser Information können die ersten beiden Schritte der übersprungenen Zyklen im jetzigen Zyklus mitausgeführt werden ( $sz - 1$  übersprungene Zyklen plus dem aktuellen Zyklus ergibt  $sz$  Zyklen !):

1.  $Z$  wird nicht um 1 Bit (Verdoppelung), sondern um  $sz$  Bits nach links verschoben und

2.  $m$  wird nicht um 1, sondern um  $sz$  erhöht.

Das Verschieben um  $sz$  Bits ist mit einem Barrel-Shifter in einem Schritt machbar (Conway; L. Mead, C., Introduction to VLSI Systems, Addison-Wesley Publishing Company, Inc., 1980).

Methoden, die es ermöglichen, überflüssige Schritte zu überspringen, werden Look-Ahead (vorausschauende) Verfahren genannt. Solche Verfahren müssen nach sorgfältiger Analyse für jeden Algorithmus getrennt entworfen werden. Es muß vor allem geprüft werden, ob der zu erwartende Zeitgewinn größer ist als die Zeit zur Berechnung der überspringbaren Zyklen. Bei der angestrebten Hardware-Implementierung des gesamten Algorithmus wird der Zeitgewinn durch nichts geschmälert, da die Berechnung der Look-Ahead-Parameter parallel zum längsten Schritt, der Addition, geschieht.

Für die Multiplikation ist seit langem ein Look-Ahead-Algorithmus bekannt. Er hat zwei Zustände:

1.  $LA = 0$ , Nullen im Multiplikator werden überlesen und

2.  $LA = 1$ , Einsen im Multiplikator werden überlesen.

Die Schrittfolge des Algorithmus lautet:

1. Setze  $sz := 1$ .

2. Setze  $m := m + 1$ .

3. Setze  $a := 1 - 2 * LA$

4. Es wird der 3-Bit-String  $M[L(M) - m, L(M) - m - 2]$  betrachtet. Solange nicht fertig, führe in Abhängigkeit vom 3-Bit-String und dem  $LA$ -Wert die in der Zeile stehende Regel aus.

$LA = 0$	$LA = 1$		
000	111	$sz := sz + 1;$	$m := m + 1.$
001	110	$sz := sz + 1;$	$m := m + 1.$
010	101	$sz := sz + 1;$	$m := m + 1.$
011	100	$LA := 1 - LA;$	Fertig. (II, III)
100	011		Fertig. (I, IV)
101	010		Fertig. (I, IV)
110	001	Unmöglich!	
111	000	Unmöglich!	

5. Im MultMod-Algorithmus ist auszuführen:

- Schiebe  $Z$  um  $sz$  Bits nach links.
- Setze  $Z := Z + a * P$ .

Die römischen Zahlen in den Klammern hinter der "Fertig"-Anweisung benennen die Regeln dieser Zeile, die erste Zahl steht für  $LA = 0$  und die zweite für  $LA = 1$ . Die 3-Bit-Strings, hinter denen "Unmöglich" steht, können nicht auftauchen, da bei ihnen bereits im Schritt davor die Regel II bzw. III zur Anwendung gekommen wäre. Die Variable " $a$ " dient nur als Zwischenspeicher für die Information, ob  $P$  beim Additionsschritt negiert wird oder nicht. In diesem Schritt findet in der Implementierung keine Multiplikation statt, da  $a$  nur die Werte  $+1$  und  $-1$  annehmen kann. Das Verschieben von  $Z$  und die Erhöhung von  $m$  ist vorher schon erklärt worden.

Die Look-Ahead-Regeln lassen sich leicht mit Hilfe der Summenzerlegung der Multiplikation verstehen. Sie lautet

$$P * M = \sum_{m=1}^{L(M)} P * M_{L(M)-m} * 2^{L(M)-m}.$$

" $s$ " soll in den folgenden Rechnungen die Stelle relativ zu  $L(M) - m$  bezeichnen, an der im Multiplikator, von der Stelle  $L(M) - m - 1$  an gerechnet, das erste Bit ungleich  $LA$  steht.

Regel I besagt, wenn in einem 0-String eine isolierte 1 steht, dann addiere an dieser Stelle  $P$  zu  $Z$ . Mathematisch ausgedrückt:

$$\sum_{\lambda=1}^{m+s} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} = \sum_{\lambda=1}^m P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} + \sum_{\lambda=m+1}^{m+s} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} = 2^{L(M)-m} * Z + 2^{L(M)-m-s} * P.$$

Die Summanden der zweiten Summe sind Null außer den Wert  $\lambda = m + s$ , da an den Binärstellen  $L(M) - m - 1 \dots L(M) - m - s$  des Multiplikator  $00 \dots 01$  steht.

Regel II ist am einfachsten im Zusammenwirken mit Regel III zu verstehen. Regel II schaltet von einem 0-String auf einen 1-String um, wenn auf die erste 1 mindestens noch eine zweite folgt. An der Stelle der letzten 0 wird  $P$  zu  $Z$  addiert. Regel III ist zu II dual. Sie schaltet von einem 1-String zu einem 0-String um, wenn auf die erste 0 mindestens noch eine zweite folgt. An der Stelle der letzten 1 wird  $P$  von  $Z$  subtrahiert. Sei  $s1$  die Stelle der ersten 1 und  $s0$  die Stelle der ersten darauf folgenden 0, beide relativ von  $m$  gerechnet, so ergibt sich:

$$\begin{aligned} \sum_{\lambda=1}^{m+s0} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} &= \sum_{\lambda=1}^m P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} + \sum_{\lambda=m+1}^{m+s1-1} P * 0 + \sum_{\lambda=m+s1}^{m+s0} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} \\ &= 2^{L(M)-m} * Z + 0 + \sum_{\lambda=m+s1}^{m+s0-1} P * 1 * 2^{L(M)-\lambda} \\ &= 2^{L(M)-m} * Z + 2^{L(M)-(m+s0-1)} * P * \sum_{\lambda=0}^{s0-s1-1} 2^{\lambda} \\ &= 2^{L(M)-m} * Z + 2^{L(M)-(m+s0-1)} * P * (2^{s0-s1} - 1) \\ &= 2^{L(M)-m} * Z + (2^{L(M)-(m+(s1-1))} - 2^{L(M)-(m+(s0-1))}) * P \end{aligned}$$

Aus der letzten Zeile folgt direkt Regel II und III. Die Stellen zwischen den beiden Umschaltpunkten brauchen nicht beachtet zu werden, vorausgesetzt alle sind 1. Ein Beispiel soll dies anschaulicher machen.

$$\begin{array}{rcl}
3 \cdot 60 & = & 180 \\
11 \cdot 00111100 & = & 10110100 \\
\begin{array}{c} \text{ } \quad \text{ } \quad \text{ } \\ \text{ } \quad \text{ } \quad \text{ } \\ \text{ } \quad \text{ } \quad \text{ } \end{array} \\
\cdot ms1 \quad s0 & L(M) = 8, m = 1, s1 = 2, s0 = 6 \\
& L(M) - m - s1 \dots L(M) - m - s2 + 1 = 5 \dots 2 \\
& L(M) - m - s1 + 1 = 6 \\
3 \cdot (32 + 16 + 8 + 4) & = & 3 \cdot (64 - 4) = 180
\end{array}$$

Man spart zwei Additionen (somit zwei Zyklen), wenn man nicht ein Bit des Multiplikators nach dem anderen abarbeitet, sondern 1-String als geometrische Summe betrachtet, die Summe berechnet und die unwesentlichen Bits überspringt.

Bleibt noch Regel IV. Sie besagt, steht in einem 1-String eine isolierte 0, dann subtrahiere an dieser Stelle  $P$  von  $Z$ . Es ergibt sich:

$$\begin{aligned}
\sum_{\lambda=1}^{m+s} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} &= \sum_{\lambda=1}^m P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} + \sum_{\lambda=m+1}^{m+s} P * M_{L(M)-\lambda} * 2^{L(M)-\lambda} \\
&= 2^{L(M)-m} * Z + \sum_{\lambda=m+1}^{m+s-1} P * 1 * 2^{L(M)-\lambda} + 0 \\
&= 2^{L(M)-m} * Z + \sum_{\lambda=m+1}^{m+s-1} P * 1 * 2^{L(M)-\lambda} + P * (2^{L(M)-m-s} - 2^{L(M)-m-s}) \\
&= 2^{L(M)-m} * Z + \sum_{\lambda=m+1}^{m+s} P * 1 * 2^{L(M)-\lambda} - P * 2^{L(M)-(m+s)}
\end{aligned}$$

Aus der Sicht des Look-Ahead-Verfahren sieht es nach der Subtraktion von  $P$  an der Stelle  $L(M) - (m + s)$  so aus, als sei der 1-String nicht unterbrochen. Der Look-Ahead kann fortgesetzt werden.

Mit dem im Flußdiagramm der Fig. 4 dargestellten Look-Ahead-Algorithmus für die Multiplikation, der die Parameter seriell berechnet, ist gegenüber der Version ohne Look-Ahead kein Zeitvorteil zu erreichen, denn pro Zyklus kann nur ein Bit des Multiplikators getestet werden. Dieses Flußdiagramm dient daher auch nur der Umsetzung der Regeln in einen funktionierenden Algorithmus. Andererseits ist dieser Algorithmus für die Hardware-Implementierung vorgesehen, jedoch geschieht dann die Berechnung der Look-Ahead-Parameter in einem Schritt parallel zu den Operationen anderer Rechenwerke, so daß am Ende eines Zyklus sofort die Parameter für den nächsten bereitstehen.

In diesem Algorithmus kann der Shift-Betrag  $sz$  maximal den Wert  $cur\_k$  annehmen ( $cur\_k$  ist der "Name" einer Variablen). Der Shift-Betrag gibt die Anzahl der Stellen an, um die ein Register verschoben wird. Ein Maximum des Shift-Betrages wird von der Theorie nicht gefordert, wohl aber von der Praxis. Der Barrel Shifter, der  $Z$  in einem Schritt in die berechnete Position schiebt, kann dies nur bis zu einem maximalen Betrag " $k$ ", der im Entwurf festgelegt werden muß.  $k$  ist der maximale Wert den  $cur\_k$  annehmen kann. Der Wert von  $cur\_k$  wird vom noch zu entwerfenden Modulo-Look-Ahead-Algorithmus festgesetzt.

Ist bis  $sz = cur\_k$  keine Regel zur Anwendung gekommen, dann wird  $Z$  um  $k$  Bits nach links verschoben und  $a$  erhält den Wert 0, d. h.  $P$  wird weder addiert noch subtrahiert. Die in einem Schritt machbare Arbeit muß aus Kosten- und Platzgründen in mehrere aufgeteilt werden. Welcher Wert für  $k$  ein günstiger Kompromiß zwischen den Forderungen der Geschwindigkeitssteigerung und der Reduzierung des benötigten Platzes ist, wird in einem der folgenden Abschnitte geklärt.

Es folgt jetzt die Look-Ahead-Erweiterung auf den Moduloschritt (Fig. 5). Da aus Gründen der Effizienz die Multiplikation in den Restklassenring verlegt wurde, nutzt die im vorigen Abschnitt gefundene Verbesserung der Multiplikation durch Look-Ahead wenig, wenn nicht auch für den Moduloschritt eine Methode zur Erzeugung von Look-Ahead-Parametern eingeführt werden kann. Sonst bremsen der Moduloschritt den gesamten Ablauf, da er nach wie vor nur ein Bit pro Zyklus abarbeiten kann. Das Verfahren ist außerdem einer Einschränkung unterworfen. Der Erwartungswert des Shift-Betrages pro Zyklus sollte annähernd übereinstimmen mit dem des Multiplikations-Algorithmus. Ist dies nicht der Fall, bremsen ein Look-Ahead-Algorithmus den anderen.

Der erfindungsgemäße Algorithmus, der die geforderten Bedingungen erfüllt, und die genannten Regeln zu einem Algorithmus zusammenfaßt, ist im Flußdiagramm der Fig. 5 dargestellt. Wie zuvor, ist auch dieser Algorithmus aus denselben Gründen seriell beschrieben. In der Hardware-Implementierung hat er dieselben Eigenschaften wie der Look-Ahead-Algorithmus der Multiplikation.

Die Rahmenbedingungen für den Modulo-Look-Ahead-Algorithmus verlangen, daß die Erwartungswerte beider Beträge, um die pro Zyklus verschoben wird, übereinstimmen. Bisher wurde  $N$  nicht verschoben:  $Z$  ist in jedem Zyklus um ein Bit nach links verschoben worden und  $N$  relativ dazu um ein Bit nach rechts, d. h.  $N$  behält seine Position bei. Dagegen wird das Look-Ahead-Verfahren einen Parameter  $sn$  generieren, der angibt, um wieviel Bits  $N$  relativ zu  $Z$  nach rechts zu verschieben ist. " $sn$ " ist im folgenden der aktuelle Shift-Betrag des Modulo-Look-Ahead-Algorithmus, und " $n$ " gibt an, wieviel Binärstellen  $N$  zum jeweiligen Zeitpunkt absolut nach links verschoben ist. Im Register  $N$  stehen demnach Vielfache des Moduls  $N$ . Deshalb wird  $Z$  am Ende der MultMod-schleife meistens kein Restklassen-Repräsentant sein. Es können drei Fälle eintreten:

1.  $sn > sz$  :  $N$  wird absolut um  $sn - sz$  Stellen nach rechts verschoben. Es muß Vorsorge getroffen werden, daß  $n$  nicht kleiner als 0 wird, da sonst mit Bruchteilen von  $N$  und nicht mit Vielfachen von  $N$  gerechnet wird. Dies zerstört die Kongruenz.  $sn$  wird deshalb so berechnet, daß der Wert von  $n$  minimal 0 wird.

2.  $sn = sz$  es ist nichts zu beachten.

3.  $sn < sz$  :  $N$  wird absolut um  $sz - sn$  Stellen nach links verschoben. Wenn die Möglichkeit besteht, daß  $n$  im nächsten Schritt einen bestimmten, eingestellten Wert  $MAX$  übersteigt, muß der Multiplikations-Look-Ahead-Algorithmus derart gebremst werden, daß  $n$  auf jeden Fall einen Wert kleiner oder gleich  $MAX$  annimmt. So wird verhindert, daß  $n$  beliebig groß wird.

Im Algorithmus von Fig. 5 ist am Schluß die Begrenzung des maximalen Shift-Betrages von  $Z$  zu sehen.  $cur\_k$  wird so gesetzt, daß im ungünstigsten Fall ( $sz = cur\_k$  und  $sn = 1$ ) des nächsten Schrittes  $n$  gerade den Wert  $MAX$  annimmt.

Ein Look-Ahead für die Modularechnung erfordert, unabhängig von den eingesetzten Regeln, daß mit Vielfachen von  $N$  gerechnet werden kann. Der Vorteil ist, daß sich die Multiplikation und die Modularechnung im allgemeinen nicht gegenseitig behindern. Eine Behinderung tritt nur dann ein, wenn im oben aufgeführten Fall 1 oder Fall 3 der Wert von  $sn$  bzw.  $sz$  für einen Schritt nach oben begrenzt wird. Ein Nachteil ist, daß die Register größere Zahlen als die des Moduls speichern müssen. Für diesen Überlauf muß im VLSI-Entwurf ein Puffer vorgesehen werden.

Dieser Nachteil gilt aber nur für zwei Register:  $Z$  und  $N$ . Außerdem ist die Puffergröße durch  $MAX$  nach oben begrenzt. Der VLSI-Entwurf kann also weiterhin von konstanten Größen aller Register und nun auch des Puffers ausgehen. Welche Größe der Puffer haben sollte, damit die Erwartungswerte nicht zu sehr sinken, wird in einem folgenden Abschnitt besprochen.

Es müssen noch die Berechnungsregeln der Look-Ahead-Parameter beschrieben werden. Dazu wird die Zahl  $ZDN$  benötigt. Sie ist definiert als zwei Drittel des Wertes vom Register  $N$ :

$$ZDN = \frac{2}{3} * N.$$

Der Algorithmus lautet:

1. Setze  $sn : = 0$ .
2. Setze  $b : = 0$ .
3. Solange der Absolutbetrag von  $Z$  kleiner oder gleich  $ZDN$  ist, führe aus:
  - a. Setze  $sn : = sn + 1$ ,
  - b. setze  $n : = n - 1$  und
  - c. schiebe  $ZDN$  um 1 Bit nach rechts, d. h. dividiere  $ZDN$  durch 2.
4. Setze  $b : = 2 * Z[\text{Vorzeichen}] - 1$ . Wenn das Vorzeichenbit den Wert 0 hat, dann ist  $Z$  positiv, andernfalls ist  $Z$  negativ.
5. Im MultMod-Algorithmus ist auszuführen:
  - a. Schiebe  $N$  um  $sn$  Bits relativ zu  $Z$  nach rechts und
  - b. setze  $Z : = Z + b * N$ , d. h. wenn  $Z$  positiv ist, dann wird  $N$  von  $Z$  subtrahiert, andernfalls wird  $N$  zu  $Z$  addiert.

Im letzten Schritt wird in der Hardware-Implementierung nicht multipliziert, da  $b$  nur die Werte  $-1, 0$  und  $+1$  annehmen kann: an die Additionslogik wird der Wert  $-N, 0$  oder  $+N$  angelegt. Die Berechnung von  $ZDN$  bereitet auch keine Schwierigkeiten, denn  $ZDN$  wird nicht jedesmal neu berechnet, sondern nur ein einziges Mal bei der Schlüsselübergabe und wird dann derselben Shift-Operation unterworfen wie  $N$ . So bleibt die Relation  $N$  zu  $ZDN$  erhalten.

Nach der Schlüsselübergabe muß  $ZDN$  jedoch berechnet werden. Zwei Drittel binär dargestellt sind 0.1010101... Die Berechnung von  $ZDN$  geschieht demnach so:

1. Setze  $Z : = 0$ .
2. Setze  $Z : = Z + N$ .
3. Schiebe  $Z$  um 2 Bits nach links.
4. Springe zu Schritt 2 zurück, wenn  $ZDN$  noch nicht genau genug berechnet ist.

Der letzte Schritt enthält eine unscharfe Abbruchbedingung.  $ZDN$  ist genau bestimmt, wenn jedes Bit des Multiplikators "Zwei Drittel" abgearbeitet ist. Die Anzahl der Bits von "Zwei Drittel", die noch einen Einfluß auf den Vergleich von  $Z$  mit  $ZDN$  haben, ist dieselbe, wie die Anzahl der Bits, die der Komparator hat, der den Vergleich durchführt. Die Breite des Komparators wird wiederum durch die verlangte Genauigkeit des Vergleiches bestimmt. Wie im nächsten Abschnitt gezeigt wird, sind 10 Bits mehr als ausreichend. Daraus folgt, daß  $ZDN$  in wenigen Schritten berechnet werden kann.

Wie gerade erwähnt, werden von  $ZDN$  nur ein paar der höchstwertigsten Bits zum Vergleich mit  $Z$  benutzt. Dies bewirkt natürlich, daß der Komparator ab und zu ein falsches Ergebnis liefert, denn ein hundertprozentig sicherer Vergleich müßte alle  $U(N)$  Bits berücksichtigen. Dies ist aus Platzgründen nur schwer zu realisieren. Was aber viel schwerer wiegt, ist die Tatsache, daß die Vergleichszeit dann ähnlich groß wird wie die normale Additionszeit. Der korrekte Vergleich wäre also ein Pyrrussieg.

Welche Auswirkungen hat es aber, wenn der Komparator eine falsche Entscheidung getroffen hat? Dann hat

$sn$  im nächsten Zyklus den Wert 1, andernfalls hätte  $sn$  einen Wert größer als 1 gehabt. Es verschlechtert sich demnach der aktuelle Shift-Betrag des nächsten Zyklus auf den Wert 1. Beweis:

Wenn der Comparator richtige Ergebnisse geliefert hat, dann ist  $sn$  so bestimmt worden, daß

$$2^{-sn} * \frac{2}{3} * N < |Z| \leq 2^{-(sn-1)} * \frac{2}{3} * N$$

ist.  $N$  wird nun um  $sn$  Bits nach rechts verschoben, d. h.  $N$  wird durch  $2$  hoch  $sn$  dividiert. Dann wird  $N$ , wenn  $Z$  negativ ist, zu  $Z$  addiert, andernfalls wird  $N$  von  $Z$  subtrahiert. Daraus folgt, daß  $N$  vom Absolutwert von  $Z$  subtrahiert wird. Das Ergebnis wird wieder in  $Z$  abgespeichert.

$$\frac{2}{3} * N - N < |Z| - N \leq \frac{4}{3} * N - N$$

$$-\frac{1}{3} * N \leq Z' \leq \frac{1}{3} * N \quad (1)$$

$$0 = > |Z'| \leq \frac{1}{3} * N.$$

Da der Absolutwert von  $Z$  jetzt kleiner gleich einem Drittel von  $N$  ist, muß im nächsten Zyklus  $sn > 1$  sein:

$$|Z| \leq \frac{1}{3} * N \Leftrightarrow |Z| \leq 2^{-1} * \frac{2}{3} * N = 2^{-1} * ZDN \Rightarrow sn > 1. \quad (2)$$

Teil 2 ist damit bewiesen.

Fällt dagegen der Komparator eine Fehlentscheidung, ist die Ungleichung 1 nicht erfüllt. Eine Fehlentscheidung wird z. B. dann getroffen, wenn  $ZDN$  durch Rundungsfehler bei der Berechnung etwas kleiner geworden ist als zwei Drittel von  $N$ . Liegt  $Z$  dann in der Nähe, aber noch unter von zwei Drittel von  $N$ , ist die Vergleichsaussage:  $ZDN$  ist kleiner als  $Z$ . Tatsächlich jedoch hätte diese Aussage erst eine Bitstelle später erfolgen müssen. Daraus folgt:

$$|Z| < \frac{2}{3} * N$$

$$Z' = |Z| - N < \frac{2}{3} * N - N = -\frac{N}{3}$$

$$\Rightarrow |Z'| > \frac{N}{3}$$

Die Voraussetzung (1) ist nicht mehr erfüllt. Es folgt

$$|Z| > \frac{1}{3} * N \Leftrightarrow |Z| > 2^{-1} * \frac{2}{3} * N = 2^{-1} * ZDN \Rightarrow sn = 1 \cdot q \cdot e \cdot d$$

Die Auswirkungen eines Fehlers des Komparators sind also relativ harmlos. Das gibt dem Designer einen weiten Spielraum in der Wahl der Komparatorweite, denn eine geringe Breite erhöht durch fehlerhafte Vergleiche nur die Zyklenzahl, verfälscht aber nicht die Rechnung. So kann ein Komparator geringer Breite entworfen werden, der zwar häufiger irrt, aber aufgrund des sehr schnell vorliegenden Ergebnisses trotzdem noch einen Geschwindigkeitsgewinn bringt.

Für diese Abwägung muß die Fehlerwahrscheinlichkeit bekannt sein. Ein Fehler ereignet sich entweder durch Rundung im letzten Bit von  $ZDN$  oder in Bits, die niederwertiger sind als das niederwertigste Komparatorbit. Sei " $d$ " die Komparatorbreite, dann ist die Fehlerhäufigkeit *Epsilon*

$$e = P(\text{Fehler}) \leq \frac{1}{2^d}.$$

Anschaulich formuliert besagt der Ausdruck, daß nur dann ein Fehler eintreten kann, wenn alle höherwertigen Bits den Vergleich nicht entscheiden konnten. Das ist bei einer von  $2^d$  Zahlen der Fall.

Da ausschließlich die wahrscheinlichkeitstheoretischen Erwartungswerte der Look-Ahead-Verfahren übereinstimmen, wurden sie notwendigerweise durch den MultMod-Algorithmus voneinander entkoppelt. Dabei wird in jedem Zyklus  $Z$  absolut und  $N$  relativ zu  $Z$  verschoben. Diese Entkopplung wird hier "Schwimmen" genannt, und nachfolgend im Zusammenhang mit Fig. 15 näher erläutert.

Der Zustand, in dem sich der Kryptographie-Prozessor jeweils befindet, ist anhand der Schritte  $a$  bis  $e$  in Fig. 15 dargestellt. Die Übergänge  $a$ ,  $c$  und  $e$  verdeutlichen den Verschiebevorgang, während  $b$  und  $d$  nicht näher erläuterte Additionen oder Subtraktionen bedeuten. Die gezeigten Rechtecke (Türme) repräsentieren die Register  $C(14)$ ,  $N(18)$  und  $Z(20, 22, 24)$ ; Register des Kryptographie-Prozessors). Die Höhe der Rechtecke beträgt 660 Bits + 20 Bits. 660 Bits ist die maximale Wortlänge, und 20 Bits ist die Größe eines Puffers, der die Entkopp-

lung ermöglicht.

Wenn beispielsweise der Schiebebetrags (Shift-Betrags; Größe der Verschiebung) der Multiplikation größer als der Schiebebetrags der Modulo-Operation ist, wird das Register  $M$  um die Differenz der Schiebebeträge zum oberen Ende geschoben (vgl. Schritte  $a$  und  $e$ ); die obersten Bits des Registers  $N$  werden somit teilweise in den Puffer hineingeschoben.

Im umgekehrten Fall wird  $N$  zum unteren Ende geschoben (Schritt  $c$ ). Die Überwachung, daß  $N$  nicht aus den Puffergrenzen hinausläuft, wird weiter unten noch näher erläutert.

Vor dem Schritt  $a$  sei  $N$  bereits um 10 Bits in den Puffer geschoben worden. Der Schiebebetrags  $sz$  nimmt dann die Werte "3", "1" und "2" an, wie unten in Fig. 15 zu erkennen ist. Dabei werden die genannten Werte in diesem Beispiel nacheinander angenommen. Der Schiebebetrags  $si$ , welcher die Verschiebung von  $N$  relativ zu  $Z$  festlegt, nimmt hier nacheinander die Werte "2", "3" und "1" an, das bedeutet, daß  $N$  absolut gesehen um  $sn = sz - si$  verschoben worden ist, also ist  $sn$  nacheinander "1", "-2" und "1". Nach den Schritten  $a$ ,  $c$  und  $e$  ist  $N$  jeweils um 11, 9 bzw. 10 Bits in den Puffer verschoben worden. Dieser Vorgang stellt das dar, was weiter oben mit dem Begriff "Schwimmen" bezeichnet wurde.

Im Schritt  $c$  ist der Einfluß der Look-Ahead-Grenze  $k$  für den Fall von  $k = 3$  verdeutlicht. Obwohl der Algorithmus für  $si$  eine Verschiebung um 4 hätte vornehmen können, ist  $N$  relativ zu  $Z$  nur um 3 Bits verschoben worden, und im Schritt  $d$  von  $Z$  weder addiert noch subtrahiert. Das bedeutet also, daß das Vorzeichen  $b$  in Fig. 5 den Wert "0" angenommen hat.

Nachfolgend wird der Verfahrensschritt der 3-Operanden-Addition erläutert. Die Fig. 6 stellt das komplette RSA-Verfahren (Fig. 6a) dem nunmehr vollständigen Ausführungsbeispiel des erfindungsgemäßen Kryptographie-Verfahrens gegenüber (Fig. 6b).

Hier sind die Abfragen, die in der letzten Fassung des MultMod-Algorithmus (Fig. 3) noch enthalten waren, durch die Aufrufe der beiden Look-Ahead-Algorithmen ersetzt. Die Berechnung der Look-Ahead-Parameter geschieht parallel. Das soll durch die parallelen Zweige, in denen die Aufrufe stattfinden, ausgedrückt werden.

In dieser Version des Algorithmus kann in  $Z$  ein negativer Wert stehen, nachdem die Schleife abgearbeitet worden ist. Deshalb muß zum Schluß im MultMod-Algorithmus mit Look-Ahead eine Ergebniskorrektur vorgenommen werden. Sollte  $Z$  negativ sein, dann ist  $Z + N$  positiv. Dieser Zusatzschritt ist im Flußdiagramm von Fig. 6b enthalten.

Der Multiplikations- und der Moduloschritt sind außerdem zu einem einzigen Schritt zusammengefaßt worden, der 3-Operanden-Addition. Es werden von der Logik pro Schritt nicht zwei, sondern drei Operanden gleichzeitig addiert, wie dies nachfolgend zu erkennen ist.

$$\begin{array}{r}
 A[i] \\
 + B[i] \\
 + C[i] \\
 \hline
 S[1] \quad S[0] = 0 \dots 3 \\
 \downarrow \quad \downarrow \\
 X[\max + 1] \dots X[i + 1] \dots 0 \\
 0 \dots Y[i] \dots Y[0]
 \end{array}$$

Die 3-Operanden-Addition wird in zwei Abschnitte unterteilt. Im ersten Abschnitt wird an jeder binären Stelle eine Summe der drei Bits der Operanden  $A$ ,  $B$  und  $C$  gebildet. Die Summe von  $A[i]$ ,  $B[i]$  und  $C[i]$  kann die Werte 0, 3 annehmen, sie ist also binär mit den zwei (1) Bits  $S[1]$  und  $S[0]$  darstellbar. Da die Summe an jeder Stelle gebildet wird, können aus den zwei Summenbits zwei neue Zahlen  $X$  und  $Y$  zusammengestellt werden ( $i = 0$  bis  $\max$ ):

$$\begin{array}{ll}
 Y[i] := S[0] & Y[\max + 1] := 0 \quad \text{und} \\
 X[i + 1] := S[1] & X[0] := 0.
 \end{array}$$

Im zweiten Abschnitt werden die beiden Zahlen auf die übliche Art und Weise addiert. Die Verlängerung um ein Bit bereitet keine Probleme, da das Ergebnis um mindestens ein Bit kürzer ist als der längste Operand.

Damit die Additionslogik nicht einen zu hohen Energieverbrauch hat, sind bei ihr an mehreren Stellen die Pullup-Transistoren weggelassen worden. Sie ist also in einem metastabilen Zustand. Kippt sie dann bei der Addierung in einen stabilen Zustand, so kann sie diesen nicht mehr selbständig verlassen. Deshalb muß die Logik am Ende eines Zyklus mit einem externen Precharge-Signal wieder in den metastabilen Anfangszustand gebracht werden. Während dieses Zeitraums wird die Bitaddition eingeschoben.

### Der Kryptographie-Prozessor

Dieser zweite Teil der Erfindungsbeschreibung befaßt sich mit dem Blockschaltbild und dem daraus resultierenden Floorplan des Prozessors.

Eine Teilaufgabe der Erfindung besteht darin, die Struktur einer spezialisierten Elementarzelle (10), die den RSA-Algorithmus optimal unterstützt, darzustellen. Mit dieser Struktur wird das Blockschaltbild des Prozessors festgelegt. Dieses enthält genügend Informationen, um einen darauf abgestimmten Floorplan des Prozessors entwerfen zu können.

Wie wird der RSA-Algorithmus effizient unterstützt? Um diese Frage zu beantworten, müssen die einzelnen

Schritte des Algorithmus auf die Eigenschaft überprüft werden, ob sie selten ausgeführt werden und/oder wenig Zeit benötigen oder ob das Gegenteil zutrifft. Im ersten Fall ist es sinnvoller, die Schritte durch Mikroprogramm zu realisieren. Bei den zeitkritischen Schritten dagegen muß die Hardware-Implementierung in die Elementarzelle verlegt werden. Es werden die folgenden Schritte auf der Ebene des RSA-Algorithmus ausgeführt (siehe Flußdiagramm in Fig. 6):

1. Die Initialisierung; den beiden Variablen  $C$  und  $e$  werden die Startwerte zugewiesen.
2. Die Schleife; in ihr gibt es zwei Abfragen und die dadurch induzierten Sprünge, zwei Aufrufe des MultMod-Algorithmus und die Inkrementierung der Variablen  $e$ .

Jeder der aufgelisteten Schritte wird entweder während der Berechnung nur einmal ausgeführt, oder die Operation beschränkt sich auf wenige Bits, z. B. die binäre Abfrage eines Exponentenbits. Außerdem sind es einfache Operationen und benötigen daher kaum Rechenzeit. Die Schritte werden also durch ein Mikroprogramm realisiert. Eine einfache Steuereinheit 42 wird dieses Problem ausführen (Fig. 11).

Die Variablen, die mit einem kleinen Buchstaben bezeichnet sind, haben alle eine Zählerfunktion. Sie arbeiten eng mit der Steuereinheit 42 zusammen, da dessen Entscheidungen von den Zählern abhängen und andererseits die Zähler in Abhängigkeit der Entscheidungen der Steuereinheit de- bzw. inkrementiert werden. Sie müssen deshalb in enger Nachbarschaft zur Steuereinheit plazierte werden. Dies ist möglich, da deren Länge einen Wert von  $\lg(L(M))$  Bits hat ( $\lg$  = Logarithmus Dualis). Bei einer Schlüssellänge von 660 Bits sind sie 10 Bits lang.

Dieselben Argumente sind für die Abfragen und die Zähler des MultMod-Algorithmus gültig. Allerdings werden diese Aufgaben von einer separaten anderen Steuereinheit 36 (Fig. 10) übernommen, da die Berechnung der Look-Ahead-Parameter sehr zeitkritisch im Hinblick auf den Ablauf der MultMod-Berechnung ist. Die rechtzeitige Generierung der Shift-Beträge  $sz$  und  $sn$  sowie der Information, ob und mit welchem Vorzeichen  $P$  und  $N$  in die 3-Operanden-Addition eingeht, beeinflusst die Zykluszeit wesentlich.

Nicht in diese Steuereinheit gehören die Operationen, die mit den  $L(N)$  Bit langen Zahlen durchgeführt werden. Für ihre Speicherung in nächster Nähe der ausführenden Logik sowie für die Logik selbst wurde eine Elementarzelle (Fig. 7) entworfen.

Die benötigten Register und die Logik der Elementarzelle 10 ergeben sich aus Fig. 7. Danach umfaßt die Elementarzelle 10 ein Register 12, welches einen Multiplikator  $M$  enthält, sowie ein Code-Register 14 und ein Datum-Register 16. Es folgt ein UD-Shift-Register 18, welches  $N$  enthält und  $-2 \dots +2$  Bit schiebt.

Weitere Bestandteile der Elementarzelle 10 sind ein Barrel-Shifter 20, ein Bitaddierer 22, sowie ein Volladdierer 24, an den sich ein Carry-Look-Ahead-Element 26 anschließt.

Im RSA- und MultMod-Algorithmus werden zusammen fünf Register benötigt:

1. Das Register 12 für den jeweiligen Multiplikator  $M$ ; die Länge des Registers ist  $L(N)$ .
2. Das Register 14 für das verschlüsselte Datum; in ihm steht während der Berechnung die Variable  $C$ , deren Wert nach Abarbeitung des RSA-Algorithmus das Ergebnis der Verschlüsselung ist. Die Länge des Registers ist  $L(N)$ .
3. Das Register 16 für das zu verschlüsselnde Datum; in ihm steht während der Berechnung die Variable  $P$ , die zu Beginn des RSA-Algorithmus den Wert des zu verschlüsselnden Datums zugewiesen bekommt. Die Länge des Registers ist  $L(N)$ .
4. Das Register 18 für den Modul  $N$ ; in ihm steht während der Berechnung ein Vielfaches des Moduls. Daher hat das Register die Länge  $L(N) + MAX$ . Dieses Register hat außer der Speicherfunktion die Fähigkeit die Variable  $N$  in einem Schritt um mehrere Stellen zu verschieben.  $N$  wird in jedem Zyklus um  $sn$  Stellen relativ zu  $Z$  nach rechts verschoben. Gleichzeitig wird  $Z$  um  $sz$  Stellen nach links verschoben, d. h.  $N$  wird absolut um  $sn - sz$  Stellen nach rechts verschoben.  $sz$  und  $sn$  können die Werte  $1 \dots 3$  annehmen (der maximale Look-Ahead  $k$  wurde auf den Wert 3 festgesetzt). Der absolute Shift-Betrag, um den  $N$  nach rechts verschoben wird, nimmt also die Werte  $-2 \dots 2$  an. Ein negativer Shift-Betrag nach rechts bedeutet, daß  $N$  nach links verschoben wird. In Anlehnung an ein LR-Shift-Register wird die geforderte Funktion mit dem UD-Shift-Register (Up/Down) 18 realisiert, das in jedem Halbzyklus in jeder Richtung um 1 Bit verschieben kann, in einem Vollzyklus also um 2 Bits.
5. Das Register  $Z$  (umfassend 20, 22, 24) für das Zwischenergebnis  $Z$  des MultMod-Algorithmus; dieses Register wird zum Beginn jedes Zyklus ausgelesen und am Ende mit dem neuen Zwischenergebnis beschrieben. Das Register muß also nur für eine kurze Zeit die Variable  $Z$  speichern. Am einfachsten wird diese Funktion durch die dynamische Speicherung jedes Bits von  $Z$  auf dem Eingangsgate eines Inverters realisiert. Die Länge des Registers ist  $L(N) + MAX$ , da das Register 18 diese Länge hat und  $Z$  dessen Wert annehmen kann. Das Register  $Z$  ist als Bestandteil des Volladdierers 24 aufzufassen.

Die drei ersten Register 12, 14 und 16 werden als statische Speicher entworfen, da sie über längere Zeiträume Informationen speichern müssen.

Neben den besprochenen Registern gehört in den Elementarzellenentwurf:

1. Ein Barrel-Shifter 20, der das Ergebnis der Addition, das Zwischenergebnis  $Z$ , um  $0 \dots 3$  Bits verschieben kann. Die Einbeziehung des 0-Bit-Shift in die Fähigkeiten des Barrel-Shifters ist notwendig, da die Multiplikation vor der Modulorechnung fertig wird und dann  $Z$  nicht (!) mehr verschoben werden darf.
2. Ein Bitaddierer 22 ohne Übertragungsbit (Carrybit), der die 3 Operandenbits in eine mit zwei Bits darstellbare Summe umwandelt.
3. Ein Volladdierer 24; so wird ein 2-Bit-Addierer bezeichnet, der das Carrybit der nächstniederen Stelle

verarbeitet und selbst ein Carrybit für die nächsthöhere Stelle generiert.

Nachfolgend wird das Multiplizieren in dem Ausführungsbeispiel erläutert. Die Additionslogik hat einen lesenden Zugriff auf die Register 16, 18 und 20, 22, 24, und sie kann in jedes Register eine Zahl abspeichern. *P* (Register 16) ist in jedem MultMod-Aufruf des RSA-Algorithmus einer der beiden Faktoren. Wird *P* immer als Multiplikant gewählt und der andere Faktor als Multiplikator, dann genügt es, wenn die Elementarzellenlogik nur die Register 16, 18 und *Z* (20, 22 und 24) auslesen kann, um die erforderlichen Shift-Operationen und die 3-Operanden-Addition ausführen zu können.

Die MultMod-Steuereinheit 36 (Fig. 10) löst die Detailaufgabe, die Look-Ahead-Parameter zu generieren. Aus diesem Grund wird der Multiplikator *M* im Register 12 parallel mit dem Register *Z* verschoben. Die MultMod-Steuereinheit 36 hat Zugriff auf die obersten Bits des Registers 12 (vgl. Fig. 12). In Abhängigkeit dieser Bits generiert die MultMod-Steuereinheit 36 den Schiebeparameter *sz*, der aber nicht direkt ausgegeben wird, sondern in entsprechende Steuersignale umgesetzt wird. Diese Vorgänge laufen in Fig. 12 in der Multiplikations-Schiebelogik 50 ab.

Entsprechend wird der Verschiebeparameter *sn* in der Modulo-Schiebelogik 52 generiert. Ein Vergleich 38 vergleicht die oberen Bits von *sz* mit  $1/3, 1/6, 1/12 \dots$  von *N*. Entsprechend dem Algorithmus aus Fig. 5 wird das Vergleichsergebnis an die Modulo-Schiebelogik 52 geliefert. Dieser Wert gibt den relativen Betrag an, um den das Register 18 relativ zu *Z* nach rechts verschoben wird. Die Modulo-Schiebelogik 52 generiert aus diesem relativen Wert und dem Schiebetrag der Multiplikations-Schiebelogik 50 den absoluten Schiebeparameter *sn*. Wiederum wird *sn* nicht ausgegeben, sondern gleich in entsprechende Steuersignale umgesetzt.

Die in Fig. 12 dargestellten Begrenzer 54 und 56 haben die Aufgabe, den Schiebetrag *sz* oder *sn* zu begrenzen, falls das Register 18 die Puffergrenzen überschreiten sollte, was weiter oben schon erwähnt wurde. Die Signale des ersten Begrenzers 54 und des zweiten Begrenzers 56 werden von der Multiplikations-Schiebelogik 50 und der Modulo-Schiebelogik 52 mit verarbeitet.

Der erste Zähler 58 in Fig. 12 enthält die Variable *m* (vgl. Fig. 6b), die angibt, wieviel Bits des Multiplikators im Register 12 noch zu bearbeiten sind. Der zweite Zähler 60 enthält die Variable *n* (vgl. Fig. 6b bzw. Fig. 5), die angibt, um wieviele Bits das Register 18 (*N*) in den Puffer hineingeschoben worden ist.

Nachdem in Fig. 7 eine Elementarzelle 10 dargestellt ist, zeigt Fig. 8 wie in einem hierarchischen Aufbau aus mehreren Elementarzellen 10 ein 4-Zellen-Block 28 mit einem hierarchischen Carry-Look-Ahead-Element 30 aufgebaut wird. Gemäß Fig. 9 sind fünf 4-Zellen-Blöcke 28 in einer weiteren Stufe zu einem 20-Zellen-Block 32 aufgebaut, und wie Fig. 10 zeigt, sind in einem weiteren hierarchischen Aufbau mehrere 20-Zellen-Blöcke 32, von denen der oberste als Puffer 34 ausgebildet ist, zu einer Verschlüsselungseinheit 40 mit einer MultMod-Steuereinheit 36 zusammengefaßt.

Eine gemäß Fig. 7 aufgebaute Elementarzelle 10 ist in der Zusammenarbeit mit der MultMod-Steuereinheit 36 in der Lage, alle Schritte der MultMod-Schleife gemäß Fig. 6b in einem Zyklus zu erledigen, da sie für jeden Schritt der Schleife (Verschieben des Modulus *N* um mehrere Bits im Register 18 (*N*); Verschieben des Registers *Z* (20, 22, 24) um mehrere Bits im Barrel-Shifter 20; Ausführung der 3-Operanden-Operation mittels des Bitaddierers 22 und des Volladdierers 24 und der Carry-Look-Ahead-Einheit 26) eine spezielle Logik enthält. Die MultMod-Steuereinheit 36 berechnet (Fig. 12) parallel zur Arbeit der Elementarzelle 10 die Parameter des nächsten Zyklus. Damit ist die direkte Umsetzung des erfindungsgemäßen Verfahrens in dem VLSI-Entwurf des Kryptographie-Prozessors gegeben.

Gemäß Fig. 13 und 14 wurde ein Carry-Look-Ahead-Element 30 entwickelt, welches erkennt, ob sich Carry-Bits über größere Distanzen beeinflussen. Da dies im Kryptographie-Prozessor bei der gewählten Lösung nur in einem von 30 000 Fällen so ist, bestimmt nicht mehr die Dauer der längsten (niederwertigste Carry-Bit beeinflusst das höchstwertigste), sondern der durchschnittlichen Additionszeit die Zykluszeit der Additionslogik.

Das Carry-Look-Ahead-Element (CLA) 30 ist hierarchisch aufgebaut. Es verarbeitet die CLA-Signale der untergeordneten Stufe (linke Seite) und generiert ein CLA-Signal für die übergeordnete Stufe (rechte Seite).

Ein Propagate-Signal einer Position bedeutet, daß der Übertrag dieser Position vom Übertrag der nächstniedrigeren Position bestimmt wird. Wenn alle Propagate-Signale aktiviert sind, generiert das hierarchische Carry-Look-Ahead-Element 30 das Propagate-Ausgangssignal dieses Elements. Ein Kill-Signal besagt, daß diese Position keinen Übertrag hat. Das Kill-Ausgangssignal wird aktiviert, wenn in den untergeordneten Elementen entscheidbar ist, daß die höchstwertige Position dieses Elements keinen Übertrag hat.

Die Carry-Look-Ahead-Elemente 30 können nach einem Baukastenprinzip zu einer Baumstruktur zusammengesteckt werden. Sie repräsentieren dann jeweils eine größere Anzahl von Carry-Bits. Der Vorteil der CLA ist, die serielle Abarbeitung der Carry-Bits durch eine parallel-serielle (baumartige) Abarbeitung zu ersetzen. Dadurch wird die Additionszeit erheblich reduziert.

In der Realität vervielfacht sich mit jeder zusätzlich benötigten Stufe allerdings die Länge der Signalfade, so daß ab einer bestimmten Baumtiefe die Zusammenfassung benachbarter Bäume keinen Gewinn mehr erbringt. Bei den Wurzeln dieser Bäume wird der Übertrag dann wieder seriell verarbeitet.

Letzteres ist in Fig. 14 zu erkennen, welche die Verschaltung der Carry-Look-Ahead-Elemente 30 in Verbindung mit einem Unterbrecher 62 zeigt. Innerhalb eines Blockes von 20 Bits werden die Überträge durch den CLA-Baum verarbeitet. Von Block zu Block wird der Übertrag seriell weitergereicht.

Dieses Konzept ist im Kryptographie-Prozessor 48 noch geringfügig erweitert. Das nutzlos gewordene Block-Propagate-Signal wird dazu verwendet, den Unterbrecher 62 zu aktivieren, der die Taktsignale für die Dauer von 8 Takten unterdrückt. Dies ist die Zeit, die ein Übertrag durch die gesamte serielle Blockkette benötigt.

Wird also der Übertrag eines Blockes vom nächstniederwertigen Block bestimmt, ist das Block-Propagate-Signal durch die Block-CLA automatisch aktiviert. Der Unterbrecher 62 wird eingeschaltet, und die Blockkette

erhält genügend Zeit, sich korrekt einzupegeln.

Die Zykluszeit kann daher so eingestellt werden, daß sie gerade ausreicht, den Übertrag direkt benachbarter Blöcke zu verarbeiten. Der Vorteil hierbei ist sehr groß, da unabhängig von der Anzahl der Stellen nur die Zeit eines Block-Übertrages berücksichtigt zu werden braucht. Die Dauer der Berechnung einer 660-Bit-Addition ist daher nicht länger als die einer 20-Bit-Addition. Lediglich in einem von ungefähr 30 000 Fällen beeinflussen sich die Carry-Bits über mehr als zwei Blöcke hinweg. Dies erkennt die CLA. Die Addition benötigt dann nicht einen, sondern acht Zyklen.

Aus der Zerlegung von RSA-Algorithmus in Grundoperationen und ihre Umsetzung in eine erfindungsgemäße Schaltung kann die absolute Schrittzahl berechnet werden, die zur Verschlüsselung eines Datums (Nachricht) notwendig ist. Daraus direkt ableitbar ist die Verschlüsselungsrate in allgemeiner Form  $V_{RSA, allg.}$  ausgedrückt in codierten Bits pro Sekunde:

$$V_{RSA, allg.} = f \cdot \frac{L(N)}{3/2 \cdot L(N) \cdot \frac{L(N)}{\min(Erw(sz), Erw(sn))} \cdot L(N) \cdot \frac{A}{B}}$$

$$= f \cdot \frac{2}{3} \cdot \frac{\min(Erw(sz), Erw(sn))}{L^2(N) \cdot \frac{A}{B}} \left[ \frac{\text{Bit}}{\text{Sekunde}} \right]$$

Sie ist

- proportional zur Frequenz  $f$  des Prozessors,
- proportional zur Anzahl der Bits  $L(N)$ , die zusammen verschlüsselt werden,
- umgekehrt proportional zur Anzahl  $3/2 \cdot L(N)$  der MultMod-Aufrufe
- umgekehrt proportional zur Anzahl  $L(N)/\min(Erw(sz), Erw(sn))$  der Additionen bzw. Subtraktionen je MultMod-Aufruf und
- umgekehrt proportional zur Anzahl  $L(N) \cdot A/B$  der Einzelschritte, in die die Addition bzw. Subtraktion einer großen Zahl zerlegt wird.  $B$  ist die Breite der ALU (je breiter die ALU ist, desto weniger Operationen sind zur Addition zweier Langwortzahlen notwendig) und  $A$  ist die Anzahl der Zyklen zur Ausführung einer Operation.

Für den neuen Kryptographie-Prozessor ist  $Erw(sz) = Erw(sn)$  und  $A/B = 1/L(N)$ , weil die Datenbreite der ALU gleich der Länge der zu verschlüsselnden Daten ist und die ALU-Operation nur einen Zyklus benötigt. Daraus folgt in diesem Fall für die Verschlüsselungsrate des Kryptographie-Prozessors  $V_{RSA, KP}$ :

$$V_{RSA, KP} = f_{KP} \cdot \frac{2}{3} \cdot \frac{Erw(sz)}{L(N)} \left[ \frac{\text{Bit}}{\text{Sekunde}} \right] f_{KP} \approx 30 \text{ MHz}, Erw(sz) \approx 2,27, L(N) = 660 \text{ Bits}$$

$$= V_{RSA, KP} \approx 68,7 \left[ \frac{\text{kBit}}{\text{Sekunde}} \right]$$

Die Frequenz von 30 MHz ergibt sich aus einer Extrapolation einer in 5 µm-NMOS-Technologie erreichten Zykluszeit von 100 ns (ein als Labormuster hergestellter Prototyp besteht aus ca. 5 000 Transistoren in 5 µm-NMOS-Technologie) auf heute gängige 2 µm-CMOS-Technologien.

Aufgebaut ist das beschriebene Ausführungsbeispiel des erfindungsgemäßen Kryptographie-Prozessors aus ca. 80 000 Transistoren in 2 µm-CMOS-Technologie. Die Chipfläche beträgt dann 5,2 mm × 5,2 mm. Bei der maximalen Schlüssel-Länge von 660 Bits ver- und entschlüsselt er im ungünstigsten Fall Daten immer noch mit einer hohen Geschwindigkeit von 64 000 Bit/sec.

Die logische Blockstruktur und der Floorplan ergeben sich aus Fig. 16 und Fig. 17. Fig. 16 zeigt das Blockschaltbild des Kryptographie-Prozessors 48, das sich unmittelbar aus den vorangegangenen Erläuterungen der Elementarzelle 10 ableiten läßt.

Bei der Konzeption des Floorplans gemäß Fig. 17 sind verschiedene Rahmenbedingungen zu beachten: Zum einen die Struktur der Elementarzelle 10, weiterhin die Kommunikation der Elementarzellen untereinander und schließlich die Gewährleistung des Anschlusses aller erforderlichen Steuersignale an die Elementarzellen.

Vier Komponenten der Elementarzelle 10 tauschen mit den entsprechenden Komponenten benachbarter Elementarzellen Information aus. Das ist erstens das UD-Shift-Register 18 (vgl. Fig. 7), zweitens der Bitaddierer 22, drittens der Barrel-Shifter 20 und viertens der Volladdierer 24. Das impliziert, daß die einzelnen Komponenten der Elementarzelle übereinander platziert werden, denn dann entstehen keine zusätzlichen Kommunikationswege zu benachbarten Elementarzellen. Aus der relativ hohen Anzahl der Komponenten folgt, daß die Elementarzelle der gewählten Ausführungsform des Prozessors eine geringe Bauhöhe und eine große Breite hat.

Obwohl die Elementarzelle flach ist, ergibt ihre notwendige Stapelung einen schmalen hohen Turm, wie links in Fig. 17 zu sehen ist. Aus fertigungstechnischen Gründen werden möglichst quadratische Chips angestrebt. Deshalb wird der Turm in einzelne Stapel aufgeteilt, die dann nebeneinander platziert werden (Fig. 17 rechts). Jeder zweite Stapel steht auf dem Kopf, weil dann die Elementarzellen, die vorher an der Trennlinie übereinander benachbart waren, zu seitlichen Nachbarn werden. Die benötigten Informationen werden an der Ober- und Unterseite der Stapel zu den Nachbarzellen übertragen.

36 31 992

Die noch nicht platzierten Einheiten des Prozessors, Hauptsteuereinheit 42 und I/O-Einheit 44, benötigen im Verhältnis zur Verschlüsselungseinheit 40 so wenig Platz, daß sie an beliebiger Stelle an ihrem Rand platziert werden können.

5

10

15

20

25

..

30

35

40

45

50

55

60

65

Nummer:  
Int. Cl.4:  
Anmeldetag:  
Offenlegungstag:

36 31 992  
G 09 C 1/00  
20. September 1986  
5. November 1987

FIG. 1

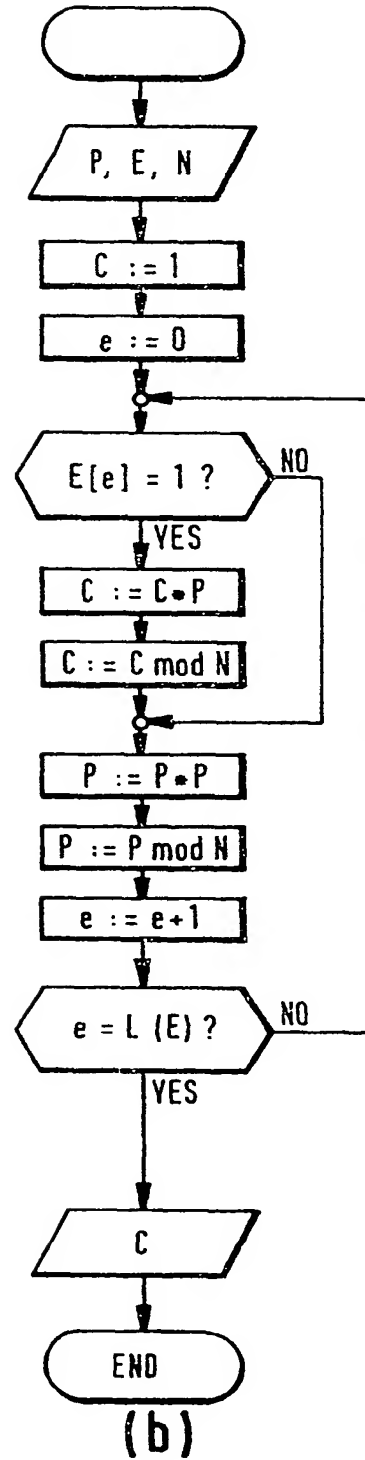
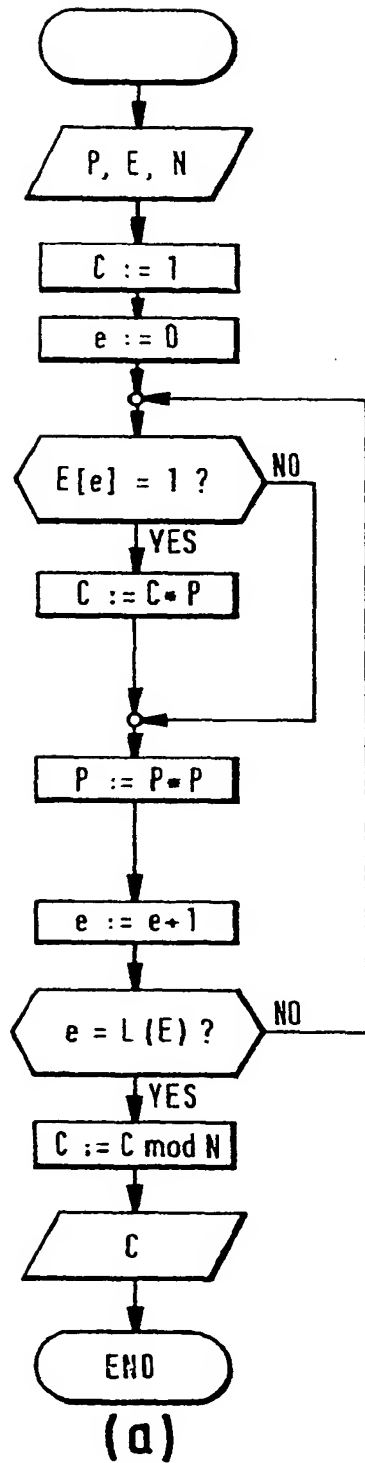


FIG. 2

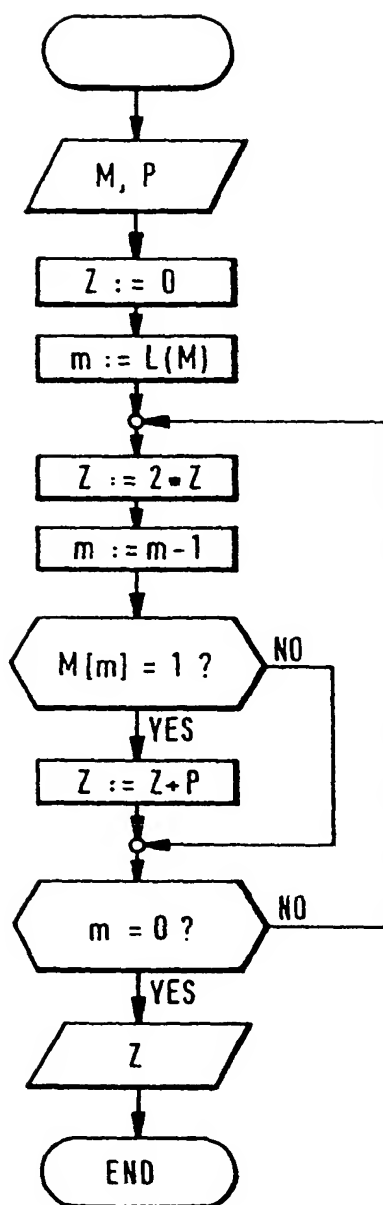


FIG. 3

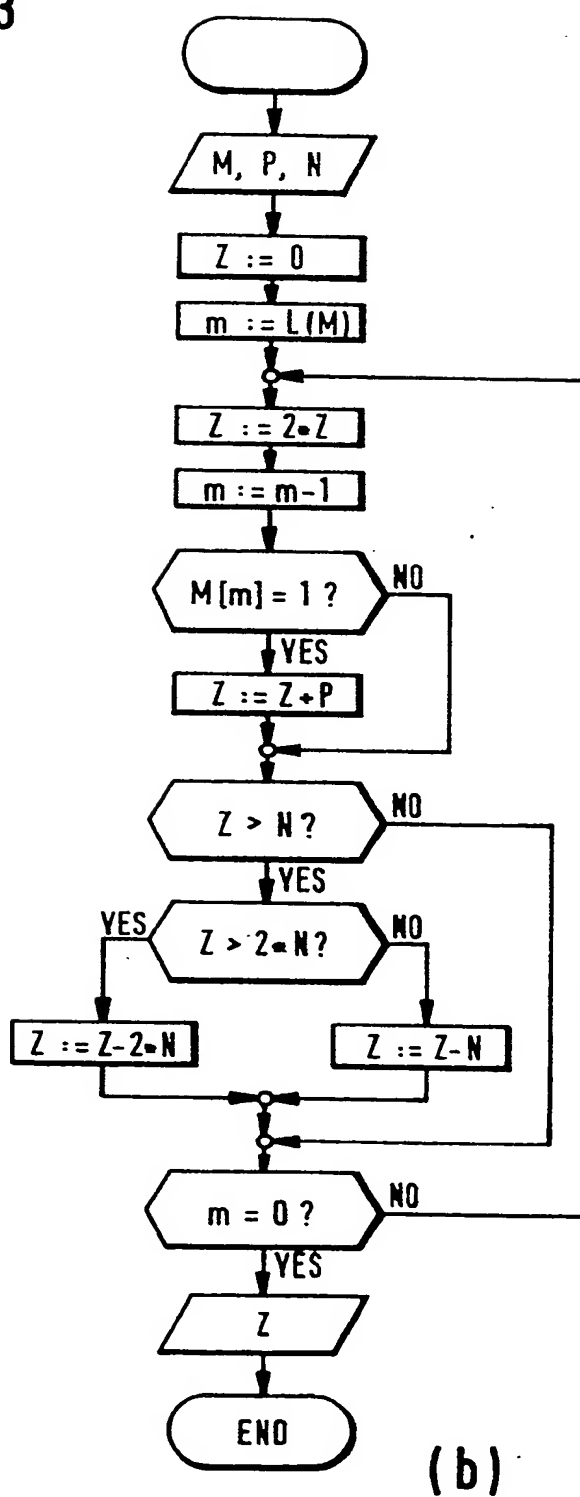
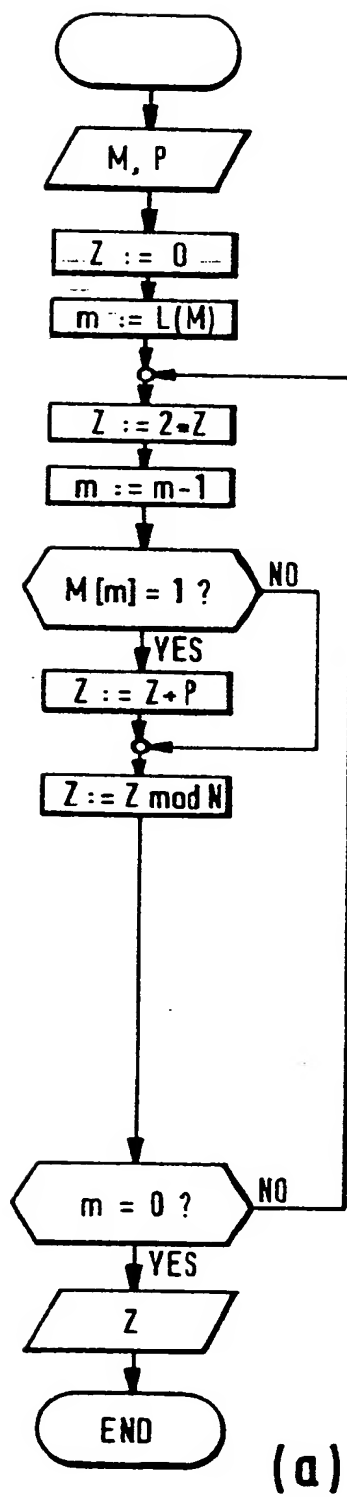


FIG. 4

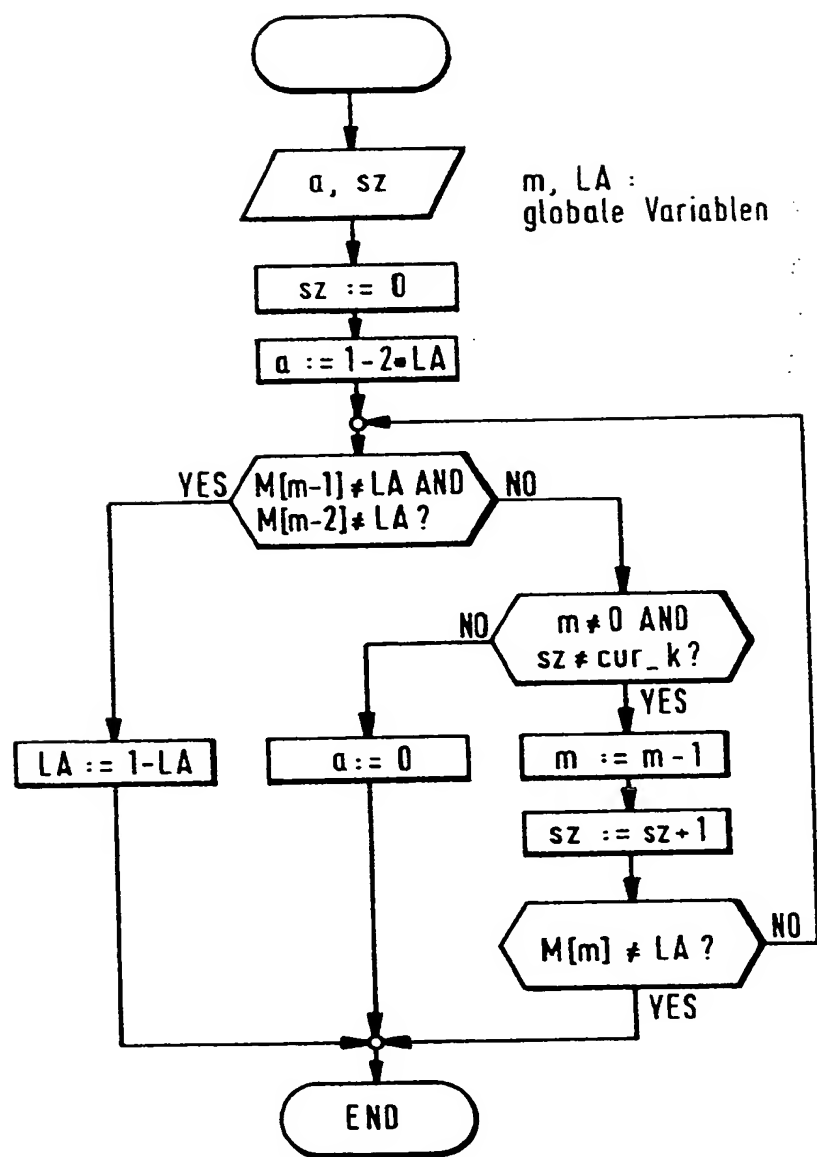


FIG. 5

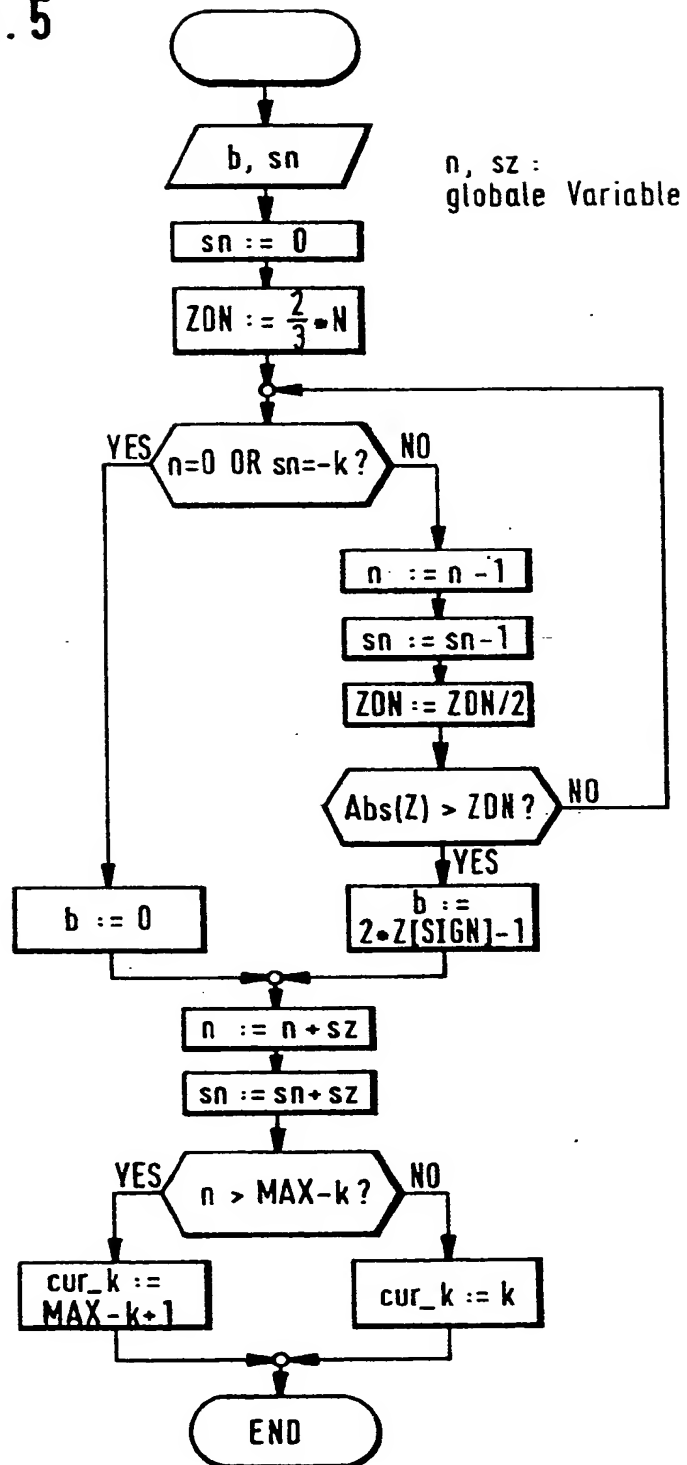
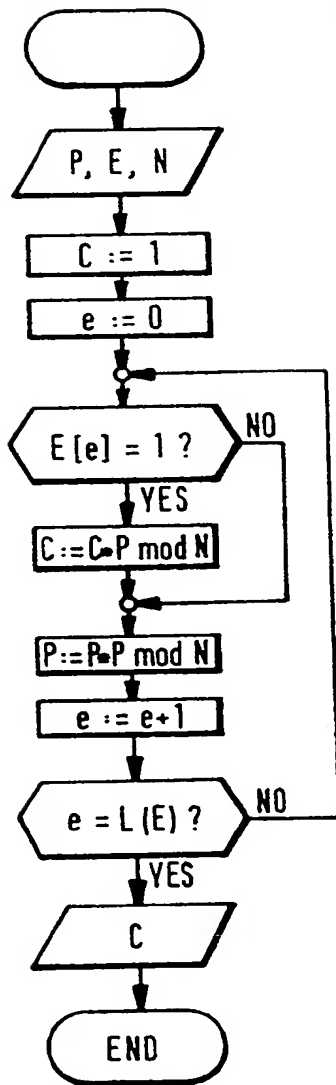
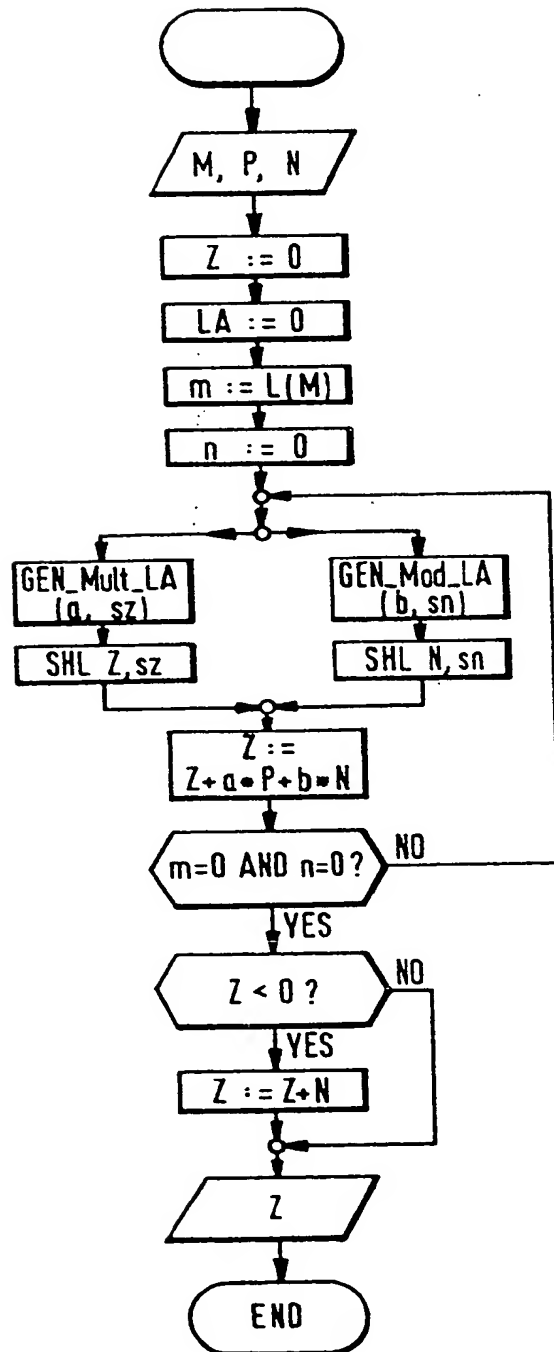


FIG. 6



(a)



(b)

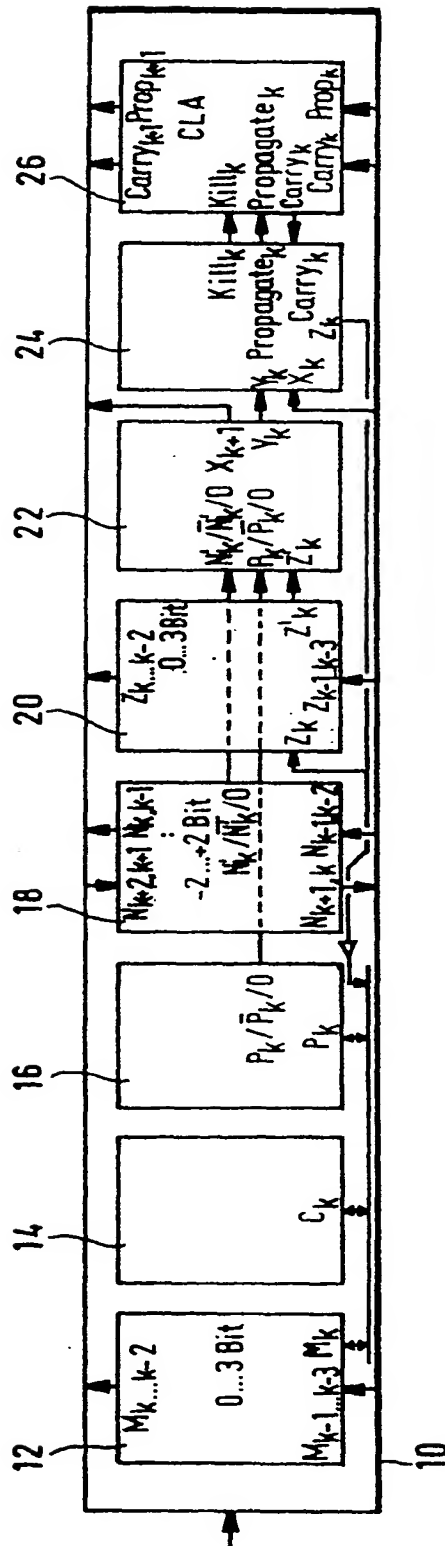
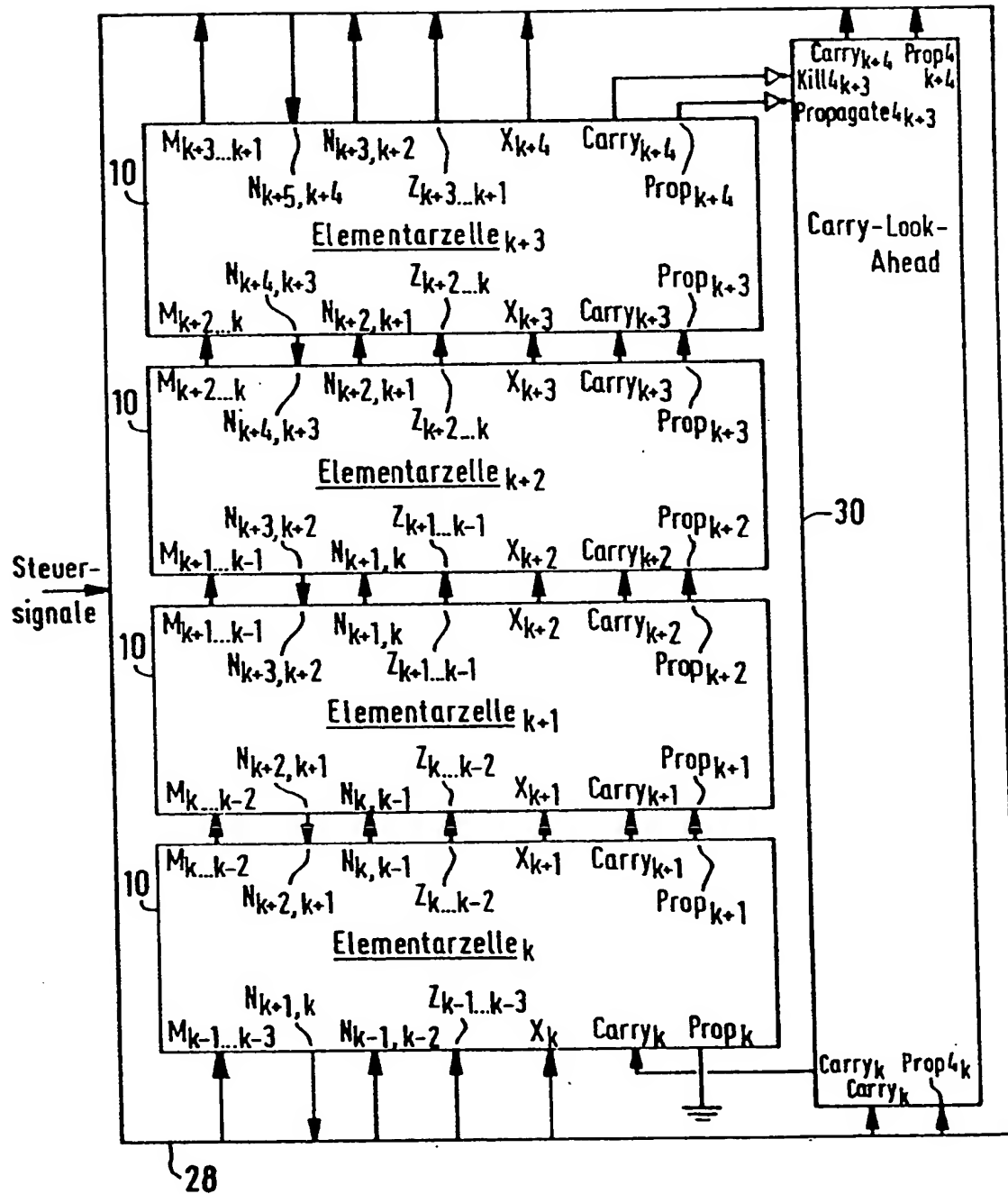


FIG. 7

FIG. 8

4-Zellen - Block  $k$ 

9/17

FIG. 9

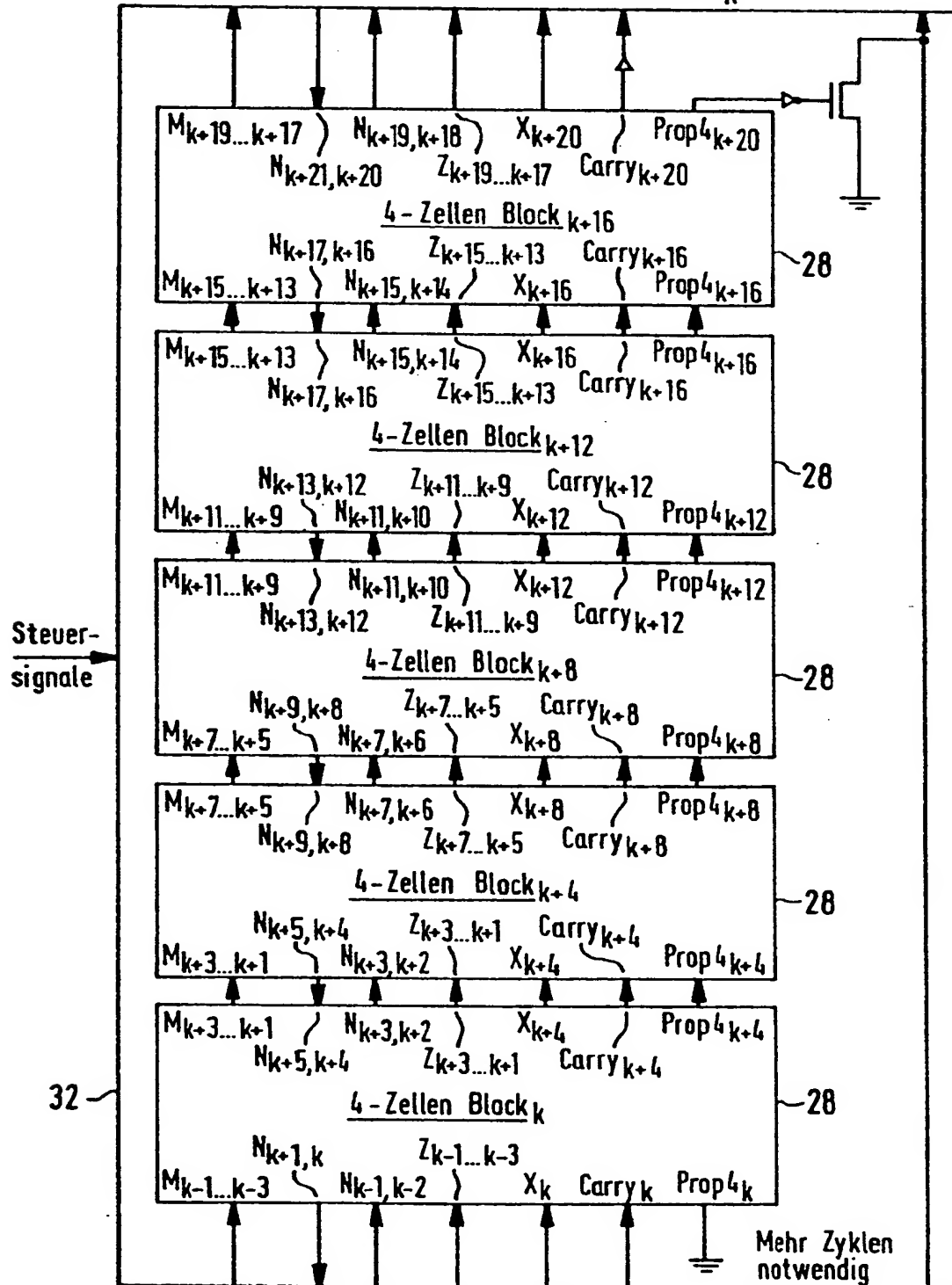
20 - Zellen Block<sub>k</sub>

FIG. 10

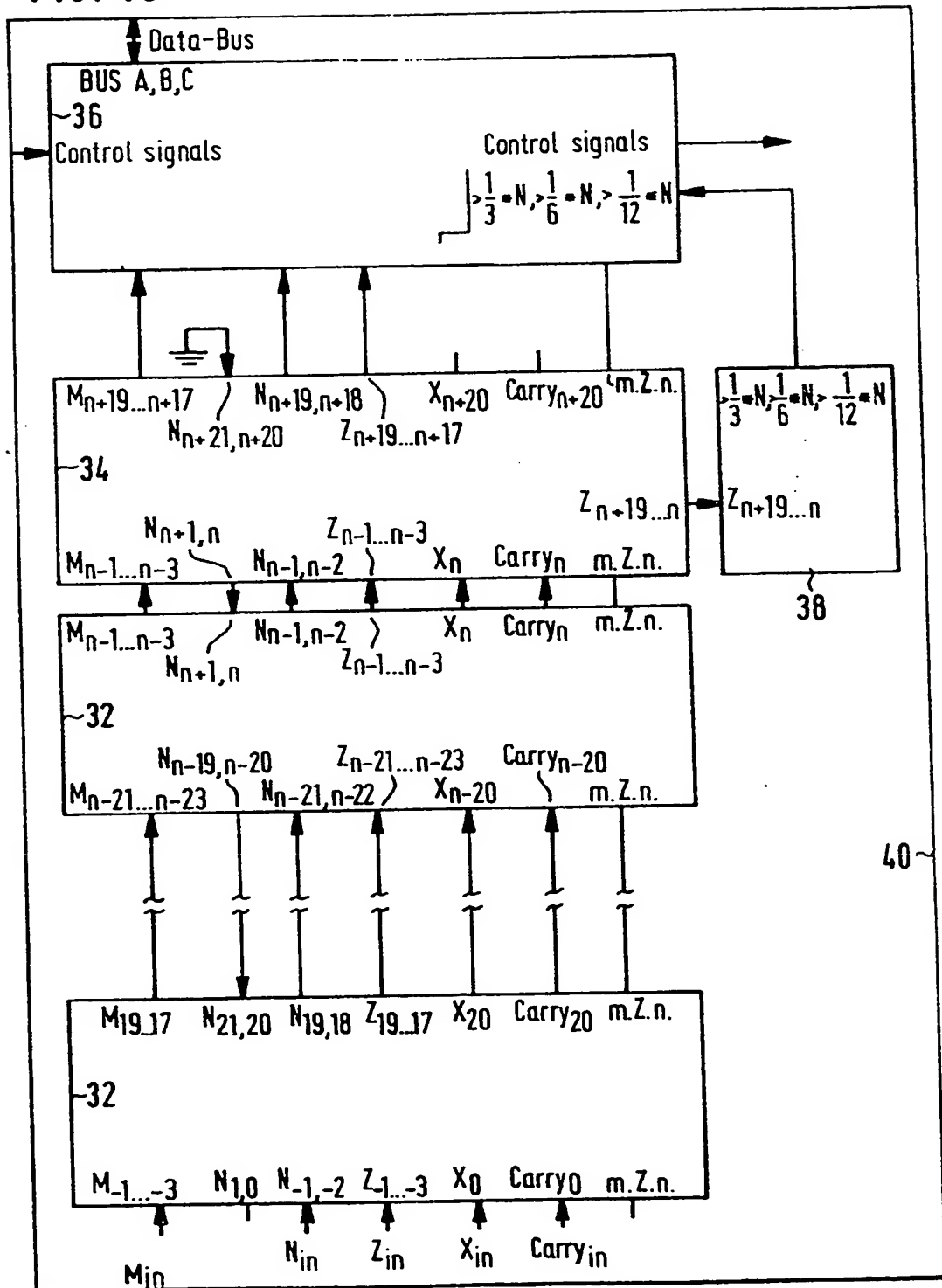
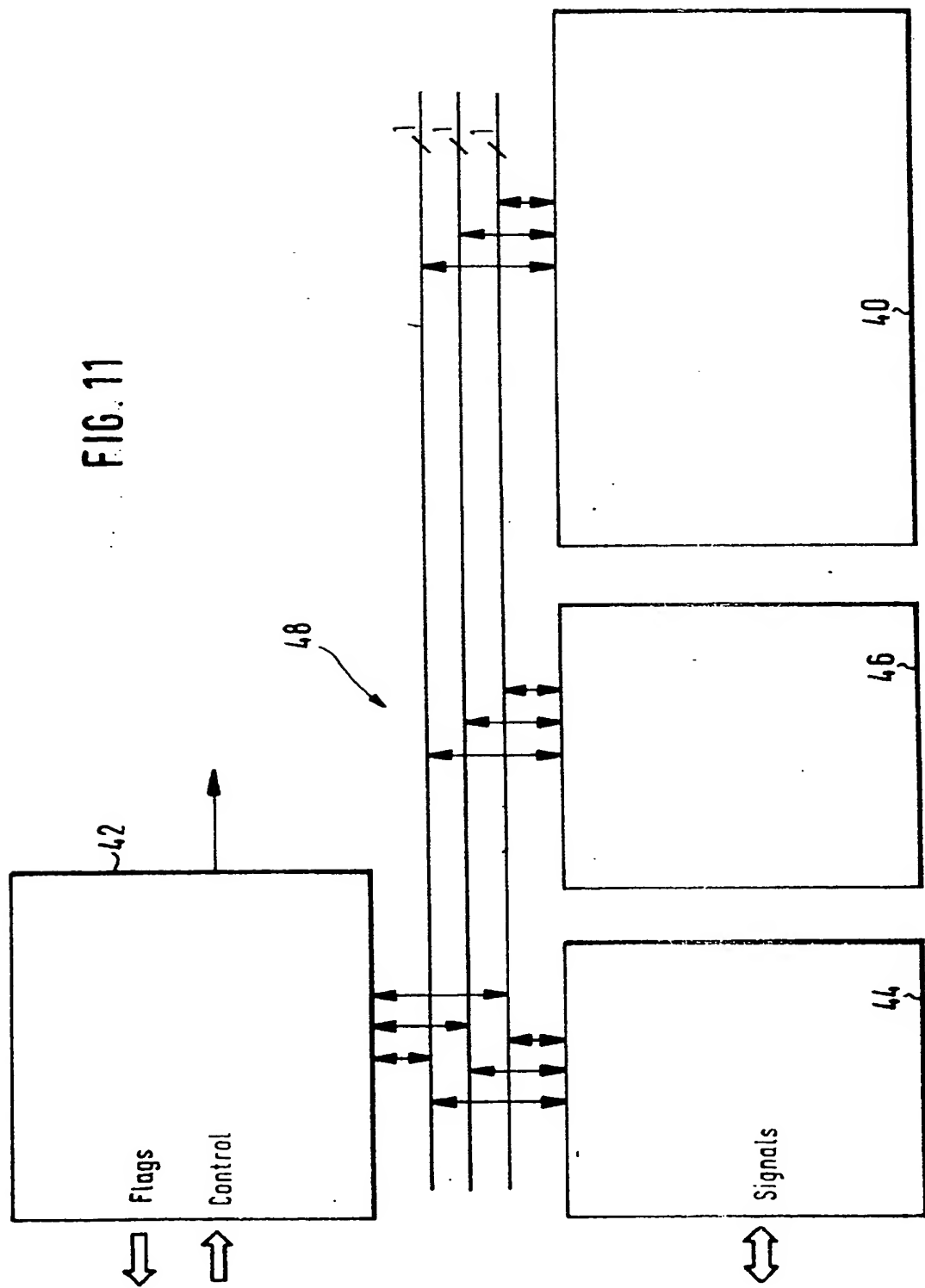


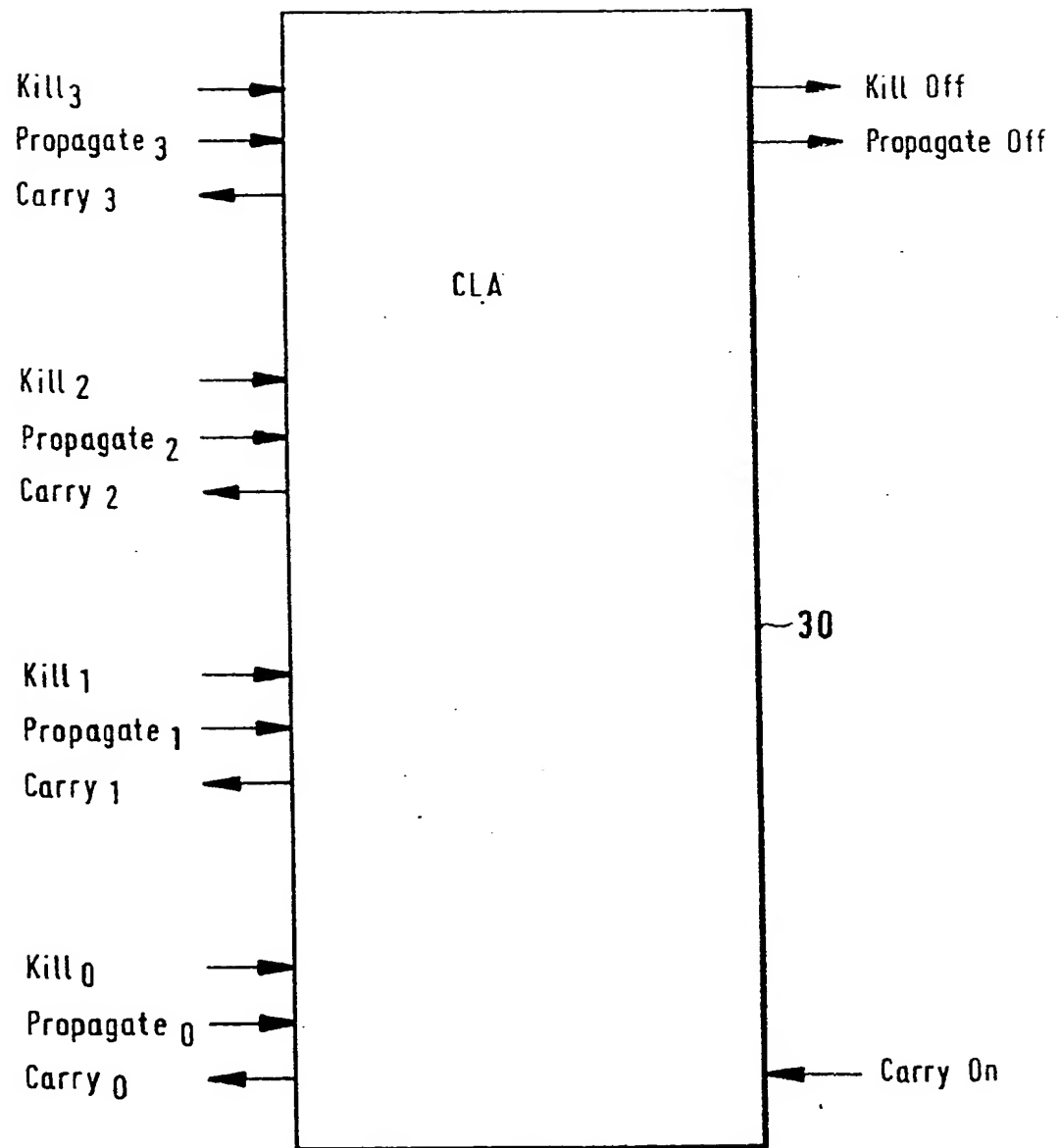
FIG. 11



ORIGINAL INSPECTED

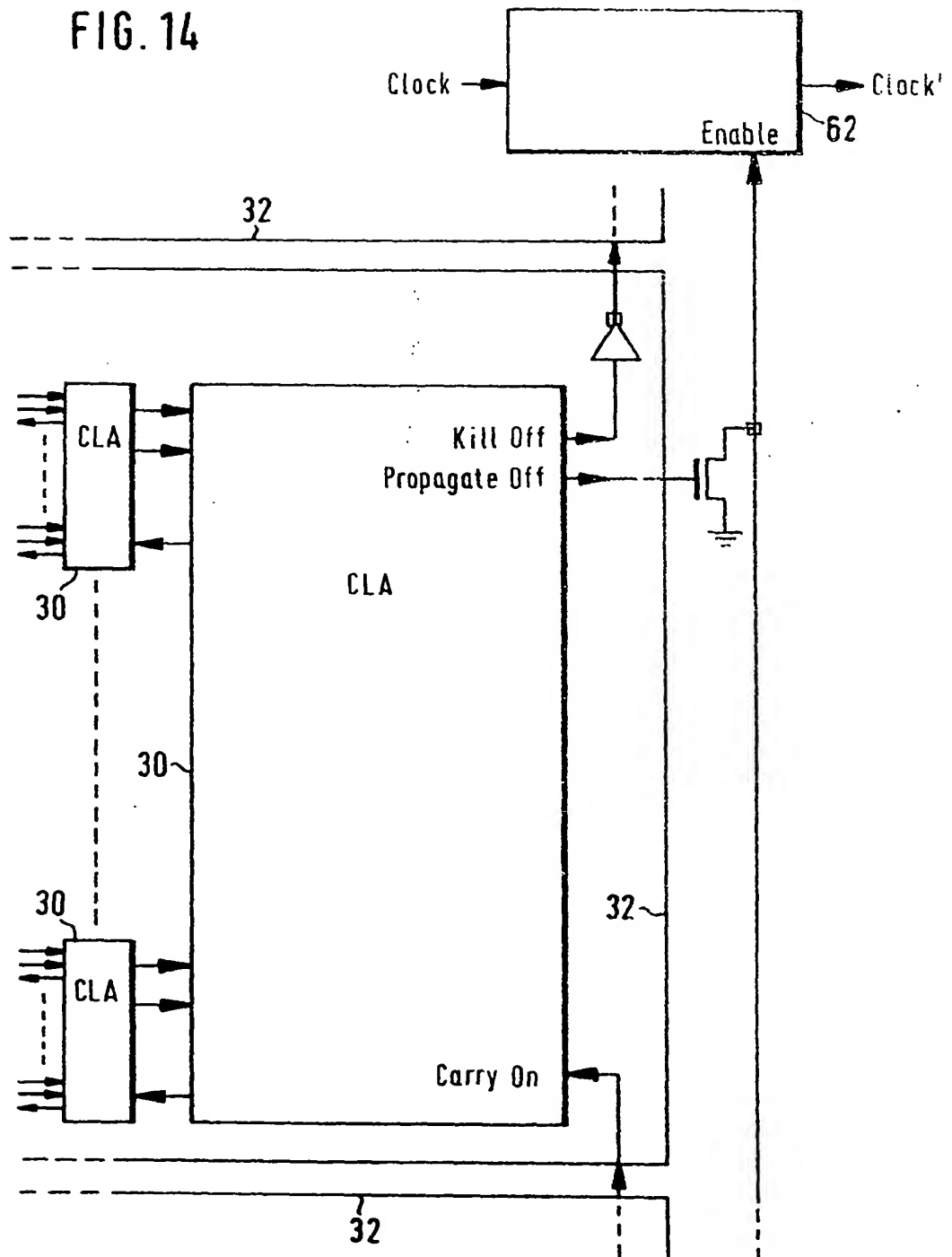


FIG. 13



ORIGINAL INSPECTED

FIG. 14



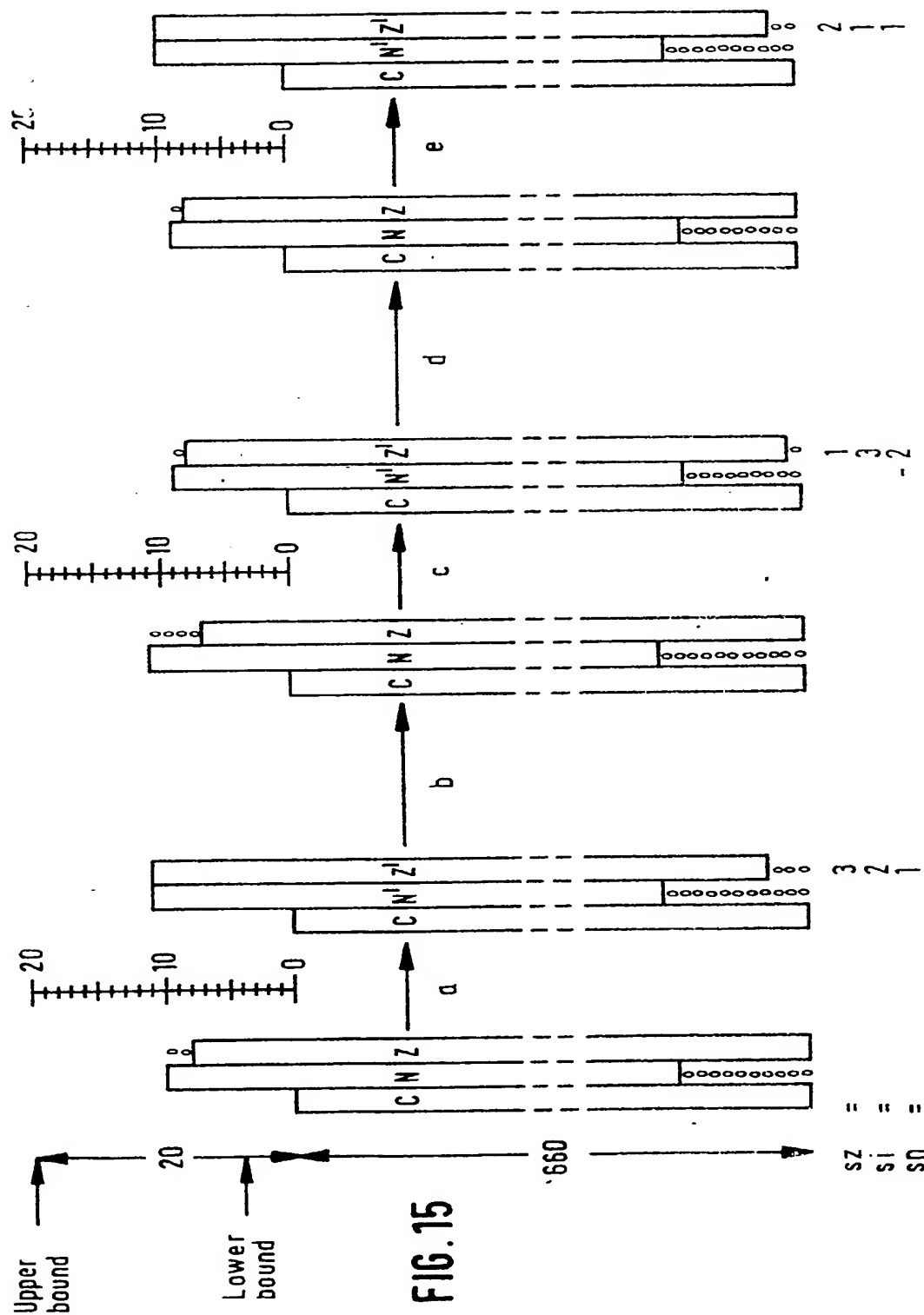
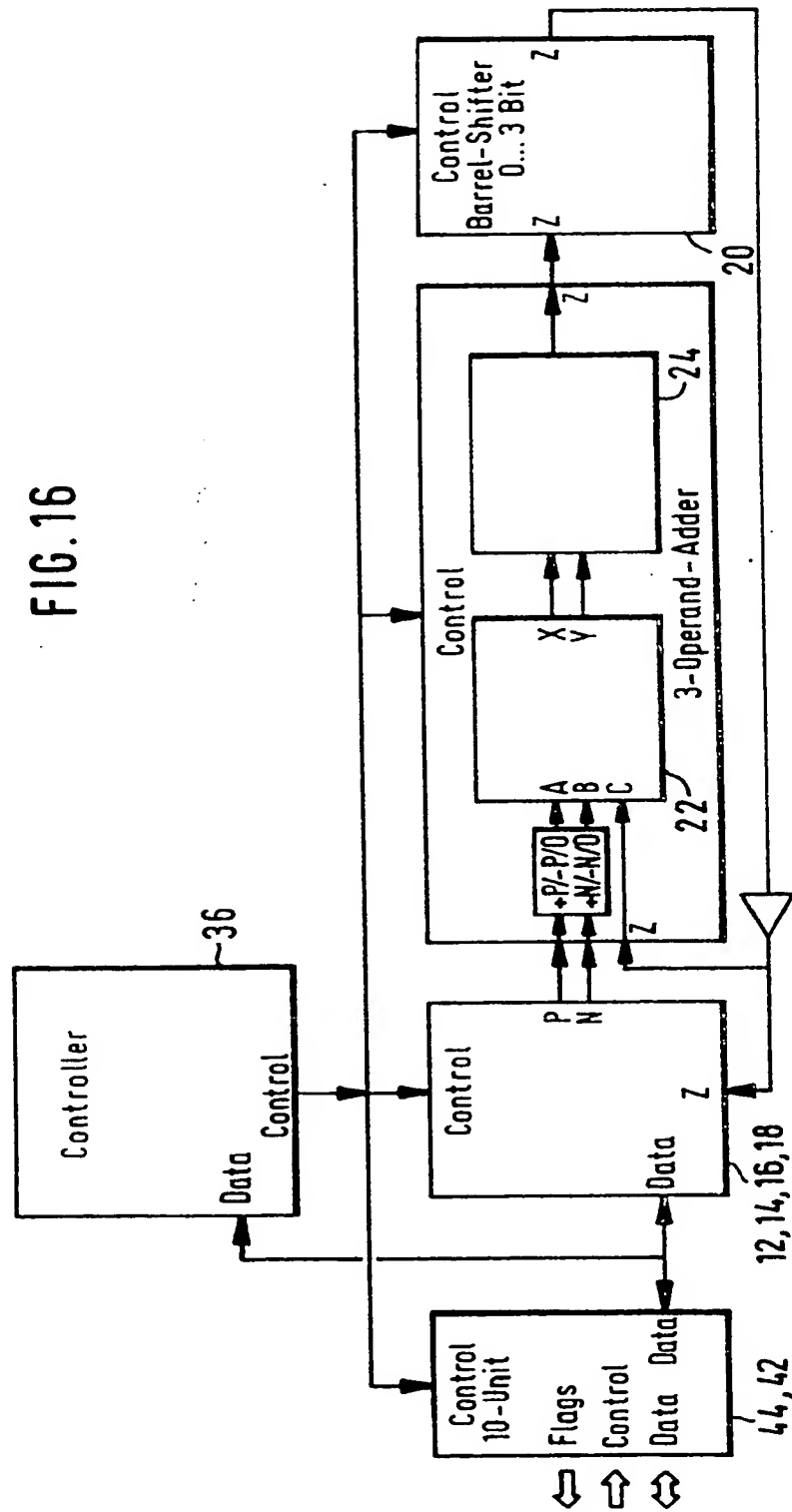


FIG. 16



DOCKET NO: S&2 TO 020103 17/17

SERIAL NO: \_\_\_\_\_

APPLICANT: Ashid Elbe et al.

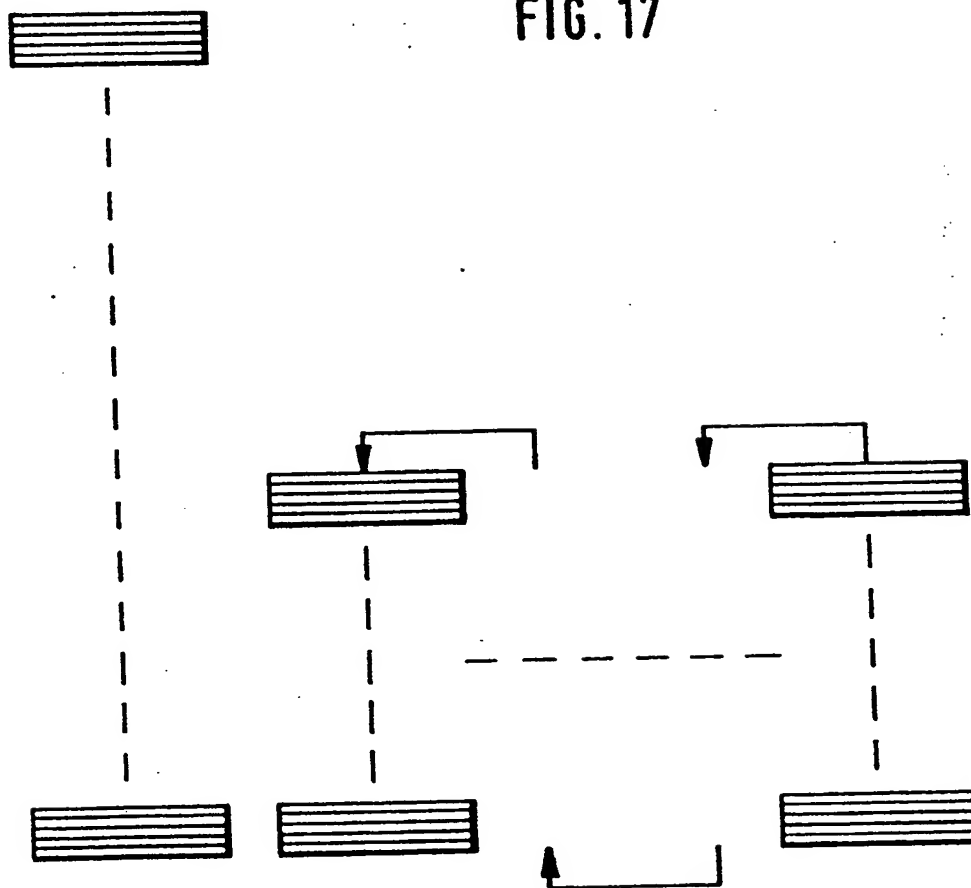
LERNER AND GREENBERG P.A.

P.O. BOX 2480

HOLLYWOOD, FLORIDA 33022

TEL. (954) 925-1100

FIG. 17



ORIGINAL INSPECTED

**Cryptography method and cryptography processor to carry out the method**

Patent Number: DE3631992  
Publication date: 1987-11-05  
Inventor(s): SEDLAK HOLGER (DE)  
Applicant(s): SEDLAK HOLGER (DE)  
Requested Patent: DE3631992  
Application Number: DE19863631992 19860920  
Priority Number(s): DE19863631992 19860920; DE19863607646 19860305  
IPC Classification: G09C1/00; G06F7/50  
EC Classification: G06F7/72A, G06F7/72C, H04L9/30F  
Equivalents:

**Abstract**

With the growing distribution of electronic methods of communication, the requirement for keeping the communication data secret and authenticating the sender is essential. The invention is based on the public key code method as implemented in accordance with the RSA method. A cryptography processor is created which encodes and/or decodes data in accordance with the RSA method, and meets the requirements of a digital interface of an ISDN network. The processor is constructed using VLSI technology, has only small dimensions for economic use, and yet makes possible encoding rates which have not previously been achieved.

Data supplied from the esp@cenet database - I2

DOCKET NO: SL2T0020103

SERIAL NO: \_\_\_\_\_

APPLICANT: Astrid Elbe et al

LERNER AND GREENBERG P.A.

P.O. BOX 2480

HOLLYWOOD, FLORIDA 33022

TEL. (954) 925-1100